

Fig. 1

User Action

200

"B" - Insert source
element before target
element 202

"A" - Insert source
element after target
element 204

"P" - Insert source
element at absolute x,y
position 206

"D" - Delete source
element 208

"R" - Replace target
element with source
element 210

"T" - replace a tribute (name
and/or value) of source element
with sNewAttr 212

"V" - replace value of source
element with sNewValue (actual
text) 214

"S" - insert source element just
after the start tag of target
element 216

"E" - insert source element just
before the end of the target
element 218

Figure 2

Generate XSLT

300

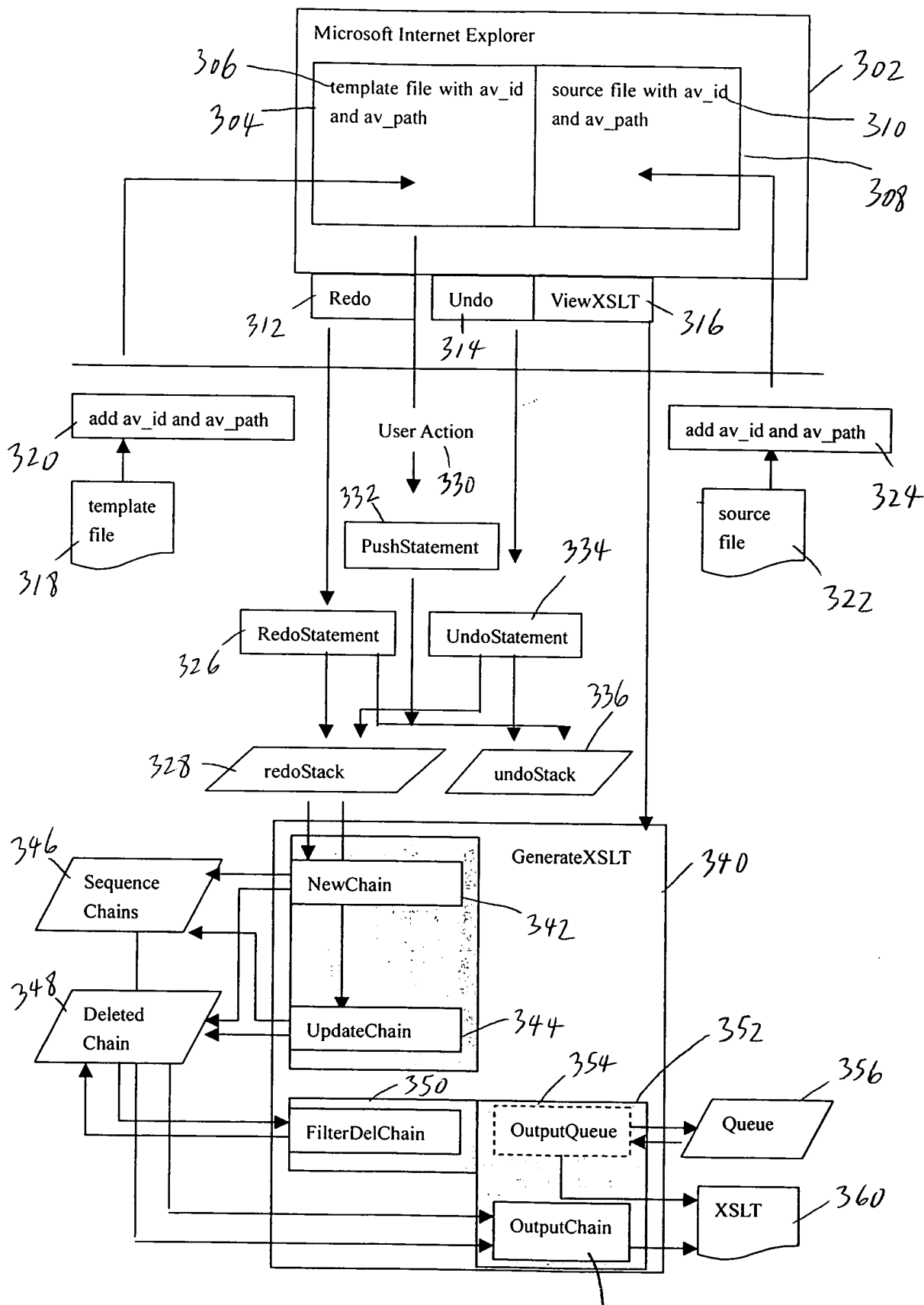


Fig. 3A

TagId

```
typedef struct _TagId{
    int      iPage;        // page /deck number
    int      iCard;        // card number for WML only
    int      iFamilyId;
    DOMString sFrame; //sample: i1.3.2
    DOMString sNewTag;
    DOMString sId;
    DOMString sName;
    DOMString sPath; //sample: tmp/html[1]/body[1]
    DOMString sAbsPos;
    bool      bIsChanged;
    bool      bIsAbsPosOrg;
    char      cAbsPos;
    int      x, y;
}TagId;
```

Fig. 4A

Statement

```
typedef struct _Statement{
    bool      bNewAction;
    char      cAction;
    struct _TagId sourceEle;
    struct _TagId targetEle;
    DOMString *psNewAttrName;
    DOMString *psNewAttrValue;
    int      iNumOfAttr;
    DOMString sNewText;
    struct _Statement *pPrev;
    struct _Statement *pNext;
}Statement;
```

Fig. 4B

Stack

```
typedef struct _Stack{
    Statement *pFirstStatement;
    Statement *pLastStatement; }Stack
```

Fig. 4C

Element

```
typedef struct _Element{
    bool    bIsChainBase;
    char     cAction;
    TagId     Ele;
    struct _Chain *pChildChain;
    struct _Attr *pFirstAttr;
    struct _Attr *pLastAttr;
    DOMString sNewText;
    struct _Element *pPrev;
    struct _Element *pNext;
}Element;
```

Fig. 4D

Chain

```
typedef struct _Chain{
    bool    bIsApplied;
    struct _Element *pChainBase;
    struct _Element *pFirstElement;
    struct _Element *pLastElement;
    struct _Chain *pPrev;
    struct _Chain *pNext;
}Chain;
```

Fig. 4E

Chains

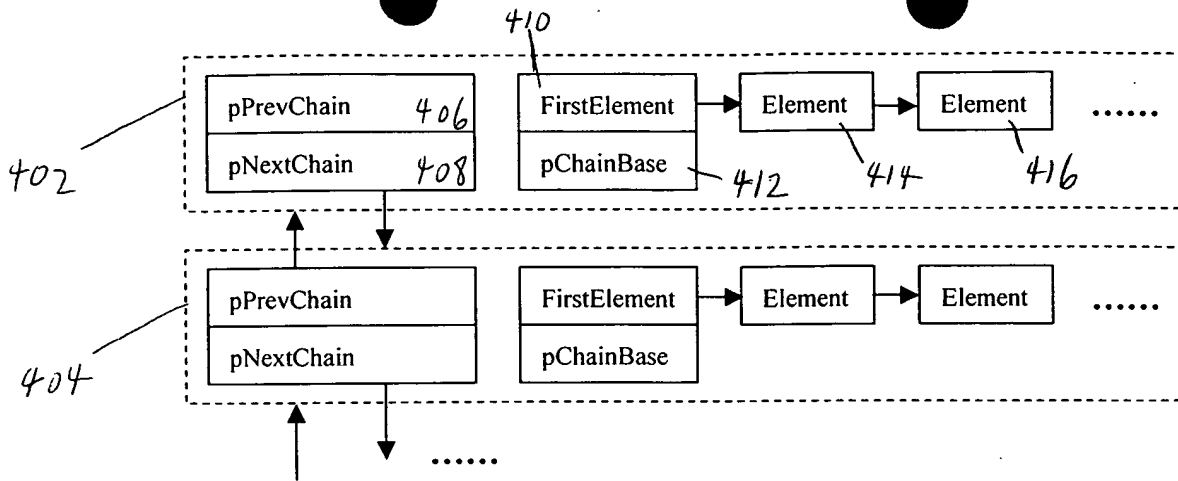


Fig. 4F

Peleted Chain



Fig. 4G

Sequence Chain



Fig. 4H

Sequence Chain with child Chain

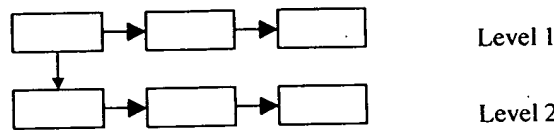


Fig. 4I

```

typedef struct _Card{
    int      iCard;
    int      iEntry;
    struct _Chain *pFirstChain;

    struct _Chain *pLastChain;
    struct _Chain DelChain;
    struct _Unit  *pFirstUnit;
    struct _Unit  *pLastUnit;
    struct _Card  *pPrev;
    struct _Card  *pNext;
}Card;

```

Fig. 4J

```

typedef struct _Page{
    int      iPage;
    struct _Card *pFirstCard;
    struct _Card *pLastCard;
    struct _Var  *pRootTmpVar;
    struct _Var  *pRootSrcVar;
    struct _Page *pPrev;
    struct _Page *pNext;
}Page;

```

Fig. 4K

```

typedef struct _Attr{
    DOMString sAttrName;
    DOMString sAttrValue;
    struct _Attr *pPrev;
    struct _Attr *pNext;
}Attr;

```

Fig. 4L


```
typedef struct _Unit{
    struct _Element    *pElement;
    struct _Unit        *pPrev;
    struct _Unit        *pNext
}Unit;
```

Fig. 4m

```
typedef struct _ElementInfo{
    bool                bSourceEleIsLocated, bTargetEleIsLocated;
    struct _TagId       sourceEle,          targetEle;
    struct _Element     *pSourceElement,    *pTargetElement;
    struct _Chain *     pChainForSourceEle, *pChainForTargetEle;
}ElementInfo;
```

Fig. 4n

```
typedef struct _Var{
    int                iMaxFrame;
    int                iMaxIFrame;
    DOMString          sFrame;
    bool               bToOutput;
    struct _Var *pPrev;
    struct _Var *pNext;
    struct _Var *pFirstFrame;
    struct _Var *pLastFrame;
    struct _Var *pFirstIFrame;
    struct _Var *pLastIFrame;
}Var;
```

Fig. 4o

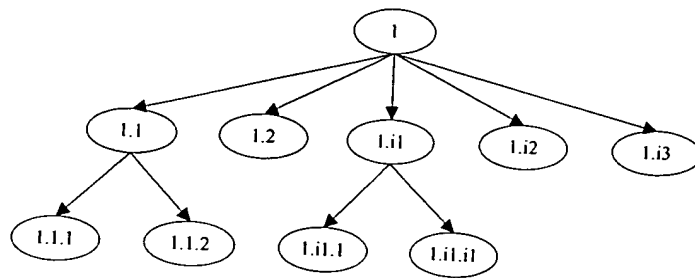


Fig. 4P

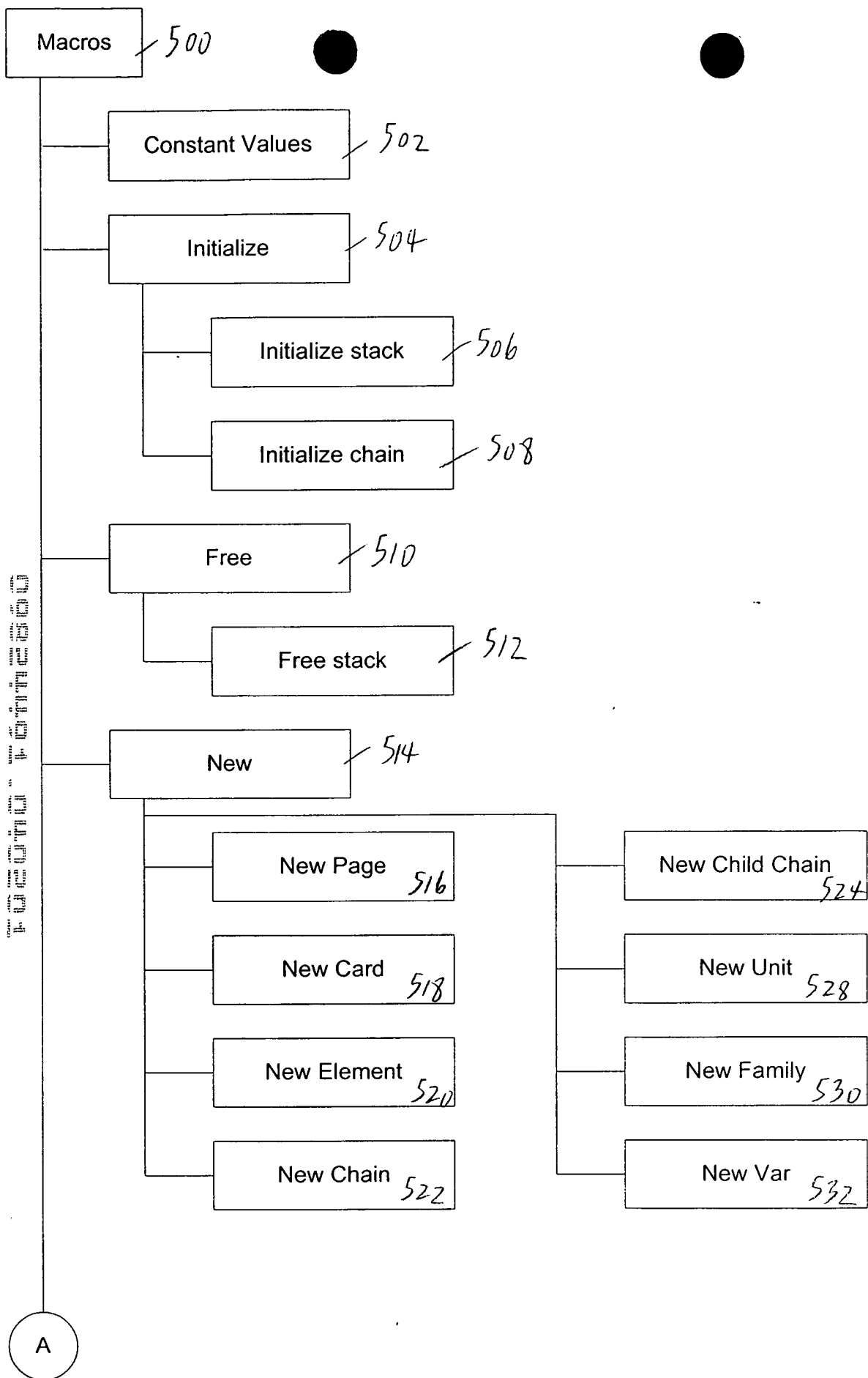


Figure 5A

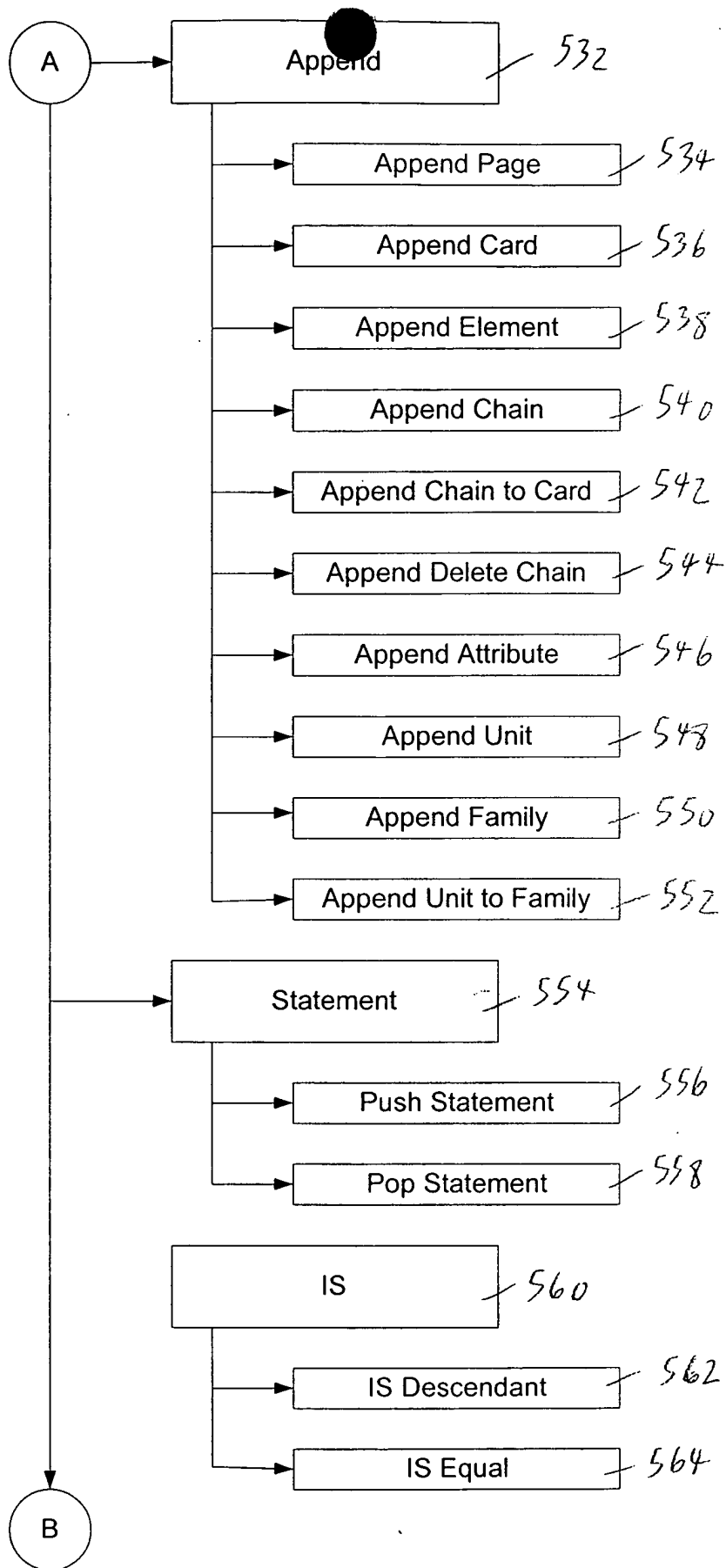


Figure 5B

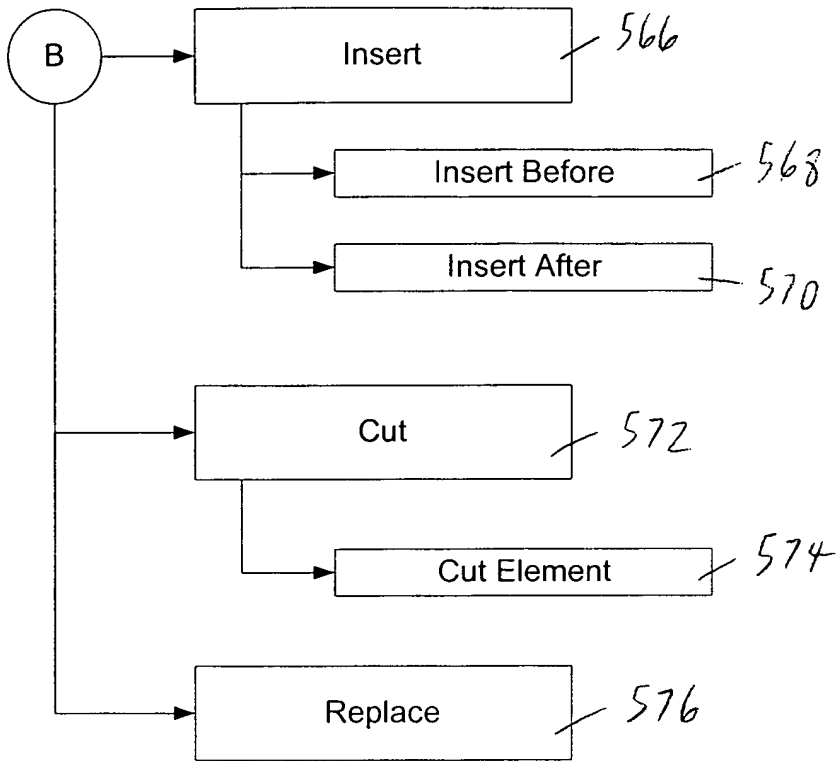


Figure 5C

Constructor

```
m_pFirstPage = NULL;
m_pLastPage   = NULL;
m_pFirstChain  = NULL;
m_pLastChain = NULL;
m_pFirstDelChain = NULL;
m_pLastDelChain = NULL;

m_bCanRedo = m_bCanUndo = false;
INIT_STACK(m_redoStack);
INIT_STACK(m_undoStack);

NEW_CHILD_CHAIN(m_pDelChain);
```

Fig. 6A

Deconstructor

```
FREE_STACK(m_redoStack);
FREE_STACK(m_undoStack);

for(pPage = m_pFirstPage; pPage != NULL; ){
    for(pCard = pPage->pFirstCard; pCard != NULL; ){
        for(pChain = pCard->pFirstChain; pChain != NULL; ){
            pChain = DeleteChain(pChain, pCard, FROM_CARD);
        }
        FREE(pPage->pFirstCard, pCard);
    }
    FREE(m_pFirstPage, pPage);
}

for(pChain = m_pFirstDelChain; pChain != NULL; )
    pChain = DeleteChain(pChain, NULL, FROM_M_PFIRSTDELCHAIN);
```

Fig. 6B

```

//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = cAction;
pStatement->sourceEle = sourceEle;
pStatement->targetEle = targetEle;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;

```

Fig. 7A

P. D.

```

//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = cAction;
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;

```

Fig. 7B

T.

```
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = 'T';
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->iNumOfAttr = iNumOfAttr;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//allocate memory for psNewAttrName and psNewAttrValue of pStatement
pStatement->psNewAttrName = new DOMString[iNumOfAttr];
pStatement->psNewAttrValue = new DOMString[iNumOfAttr];

//set psNewAttrName and psNewAttrValue of pStatement
for(i=0;i<iNumOfAttr;i++){
    (pStatement->psNewAttrName)[i] = psNewAttrName[i];
    (pStatement->psNewAttrValue)[i] = psNewAttrValue[i];
}

//Push into redoStack
PUSH_STATEMENT(m_redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;
```

Fig. 7C

V.

```
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = 'V';
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->sNewText = sNewText;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(m_redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;
```

Fig. 7D

Undo Statement

```
//pop out from redoStack  
POP_STATEMENT(m_redoStack, pStatement);  
  
//push into undoStack  
PUSH_STATEMENT(m_undoStack, pStatement);  
  
//set redo/undo  
m_bCanRedo = true;  
if(m_redoStack.pFirstStatement == NULL)  
    m_iCanUndo = false;  
return;
```

Fig. 8A

Redo Statement

```
//pop out from undoStack  
POP_STATEMENT(m_undoStack, pStatement);  
  
//push into redoStack  
PUSH_STATEMENT(m_redoStack, pStatement);  
  
//set redo/undo  
m_bCanUndo = true;  
if(m_undoStack.pFirstStatement == NULL)  
    m_iCanRedo = false;  
return;
```

Fig. 8B

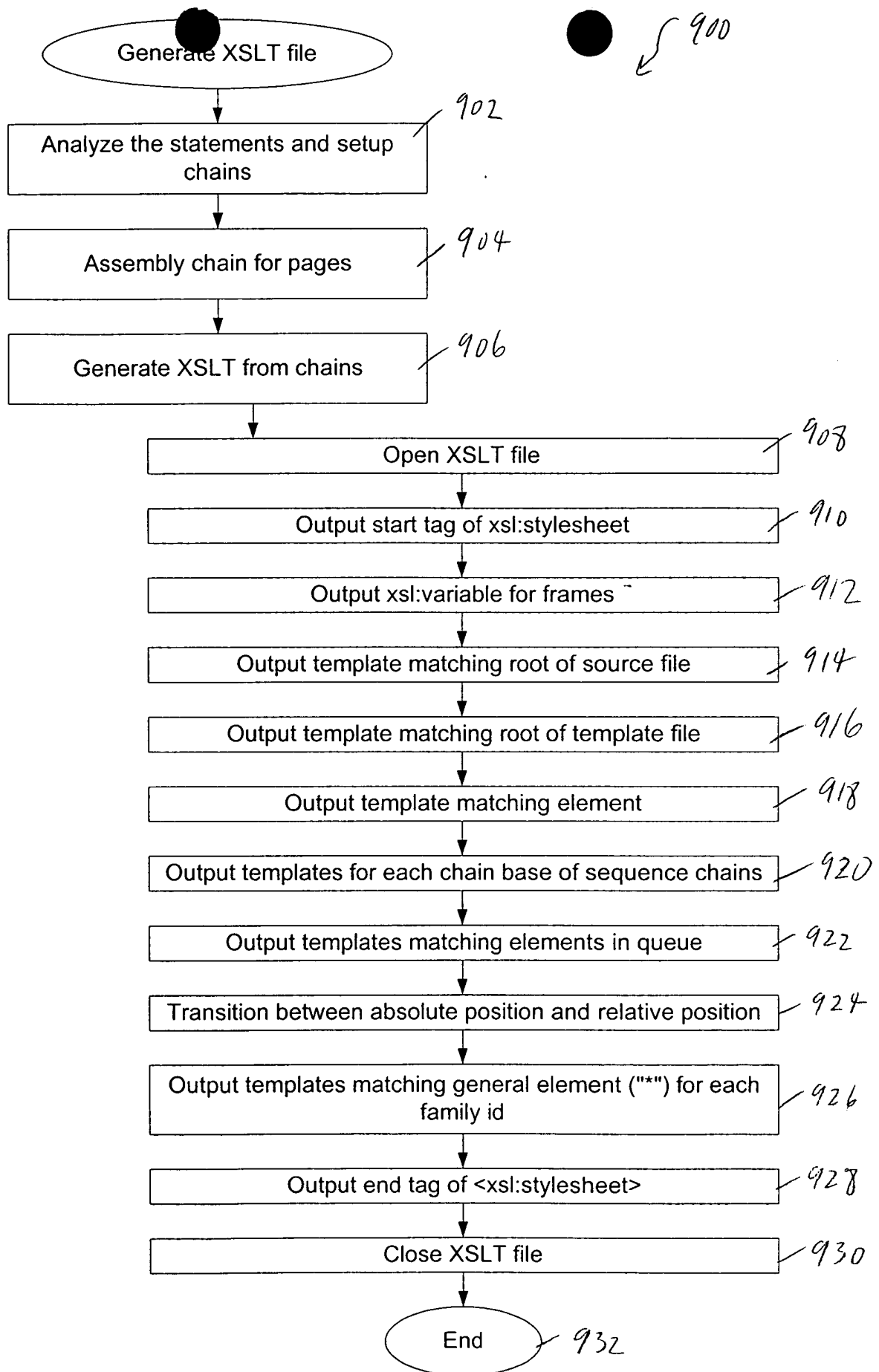


Fig. 9A

Generate XSLT

Step. 1

//step 1.1: loop from the first statement to the last statement to assembly sequence chains and deleted chains

pCurrStatement = m_redoStack.pFirstStatement;

while(pCurrStatement != NULL){

 //step 1.1.1: assembly elementInfo

 elementInfo.sourceEle = pCurrStatement->sourceEle;

 elementInfo.targetEle = pCurrStatement->targetEle;

 elementInfo.pSourceElement = NULL;

 elementInfo.pChainForSourceEle = NULL;

 elementInfo.pTargetElement = NULL;

 elementInfo.pChainForTargetEle = NULL;

 //iPage equals 0 means that the element is from source file

 if(elementInfo.sourceEle.iPage == 0)

 elementInfo.bSourceEleIsLocated = true;

 else

 elementInfo.bTargetEleIsLocated = false;

 //targetEle's iPage is impossible to be = 0

 elementInfo.bTargetEleIsLocated = false;

 //step 1.1.2: Check whether sourceEle and targetEle are in sequence chains and call methods.

 LocateElement(m_pFirstChain, &elementInfo);

 //if targetEle is not in any sequence chain, call NewChain, otherwise call UpdateChain

 if(elementInfo.pTargetElement == NULL)

 NewChain(&elementInfo, pCurrStatement);

 else{

 UpdateChain(&elementInfo, pCurrStatement);

 }

 //step 1.1.3: Transit to the next statement

 pCurrStatement = pCurrStatement->pNext;

}//end of while

//step 1.2: Filter the deleted chain

FilterDelChain(pPage);

Fig. 9B.

Generate XSLT

Steps 3.4 - 3.7

Step 3.4. Output template matching root of source file

Format

```
<xsl:template match="/">
  <xsl:apply-templates select="$tmp/*" mode="tmp_root_output"/>
</xsl:template>
```

Step 3.5. Output template matching root of template file

Format

```
<xsl:template match="*" mode="tmp_root_output">
  <xsl:copy>
    <xsl:for-each select="@*"><xsl:copy/></xsl:for-each>
    <xsl:apply-templates select="//comment()" />
    <xsl:apply-templates select="*|text()" mode="tmp_test_# " />
    ....
  </xsl:copy>
</xsl:template>
```

Fig. 9C

Step 3.6. Output template matching comment

Step 3.7. Output templates for each chain base of sequence chains

Format

```
<xsl:template match="chain base's path" mode="XY_base_##">
  <xsl:apply-templates select="the path of the Element in the chain" mode="x_output_#"/>
  ....
</xsl:template>
```

Fig. 9D

Generate XSLT

Step. 3.7

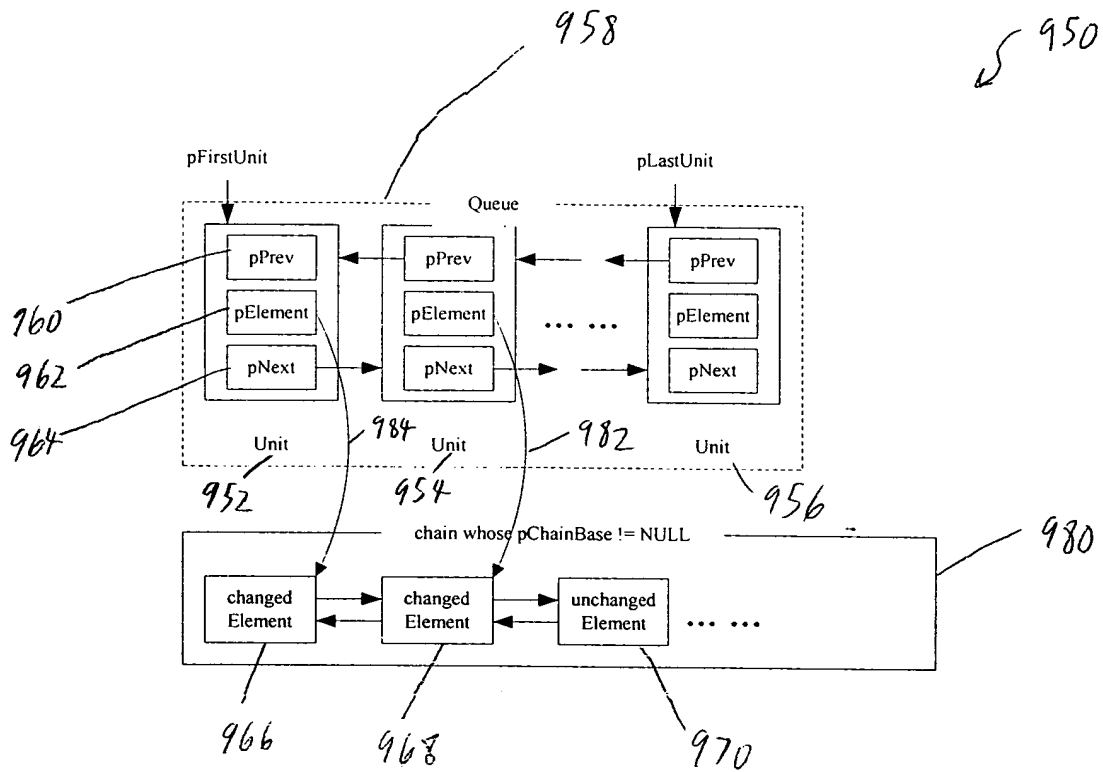


Fig. 9E

Private
Methods

1000

Destroy Element

1002

Delete Element

1004

Delete Chain

1006

Locate Element

1008

New Chain

1010

New Child Chain

1012

Update Chain

1014

Update Del Chain

1016

Assembly Chain

1018

Parse Frame

1020

Output Var

1022

Output Chain

1024

Get Unit

1026

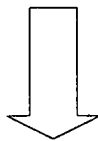


Figure 10

Destroy element

```
if(pElement->pChildChain) DeleteChain(pElement->pChildChain);  
free(pElement);
```

Fig. 11A

Delete element

Step 1. Cut the *Element* from the chain.

```
pNext = pElement->pNext;  
CUT(pChain->pFirstElement, pChain->pLastElement, pElement);
```

Step 2. Destroy it.

```
DestroyElement(pElement);
```

Step 3. Finally return the pointer to the next *Element*.

```
return pNext;
```

Fig. 11B

delete chain

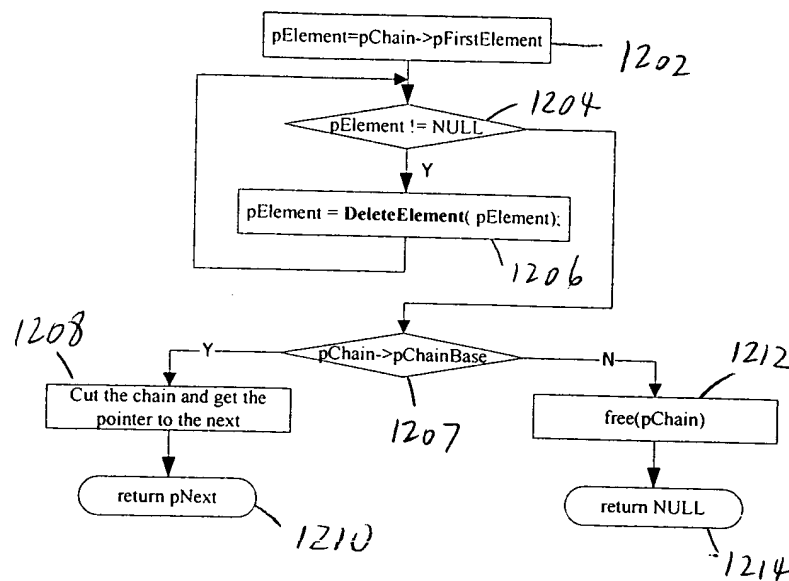


Fig. 12A

Pseudo code

```

pElement = pChain->pFirstElement;
//delete all Elements in this chain
while(pElement){
    pElement = DeleteElement(pChain, pElement);//return the next Element.
}
//delete the chain
if(pChain->pChainBase == NULL){//it is a child chain or Deleted Chain
    free(pChain);
    return NULL;
}
else{
    free(pChain->pChainBase);
    pNext = pChain->pNext;
    switch(mode){
        case FROM_M_PFIRSTCHAIN:
            CUT(m_pFirstChain, m_pLastChain, pChain);
            break;
        case FROM_CARD:
            CUT(pCard->pFirstChain, pCard->pLastChain, pChain);
            break;
        case FROM_M_PFIRSTDELCHAIN:
            CUT(m_pFirstDelChain, m_pDelLastChain, pChain);
            break;
    }
    free(pChain);
  
```

Fig. 12B

Locate Element (1)

```

sourceEle = pElementInfo->sourceEle;
targetEle = pElementInfo->targetEle;
bSourceEleIsLocated = pElementInfo->bSourceEleIsLocated;
bTargetEleIsLocated = pElementInfo->bTargetEleIsLocated;
//loop for all sequence chains
for(pRefChain = pChain; pRefChain != NULL; pRefChain = pRefChain->pNext){
    //set bToLocateSourceEle and bToLocateTargetEle
    if(pRefChain->pChainBase){
        if(bTargetEleIsLocated == false &&
            pRefChain->pChainBase->Ele.iPage == targetEle.iPage &&
            pRefChain->pChainBase->Ele.iCard == targetEle.iCard)
            bToLocateTargetEle = true;
        else
            bToLocateTargetEle = false;

        if(bSourceEleIsLocated == false &&
            pRefChain->pChainBase->Ele.iPage == sourceEle.iPage &&
            pRefChain->pChainBase->Ele.iCard == sourceEle.iCard)
            bToLocateSourceEle = true;
        else
            bToLocateSourceEle = false;
    }
    else{
        bToLocateSourceEle = !bSourceEleIsLocated;
        bToLocateTargetEle = !bTargetEleIsLocated;
    }
    //if bToLocateSourceEle or bToLocateTargetEle is not true, search in this chain
    if(bToLocateSourceEle || bToLocateTargetEle){
        //loop for all Elements in this chain
        pRefElement = pRefChain->pFirstElement;
        for( ; pRefElement; pRefElement = pRefElement->pNext){
            if(bToLocateSourceEle){
                if(IS_EQUAL(pRefElement->Ele, sourceEle)){
                    pElementInfo->pSourceElement = pRefElement;
                    pElementInfo->pChainForSourceEle = pRefChain;
                    pElementInfo->bSourceEleIsLocated = true;
                    bSourceEleIsLocated = true;
                }
            }
        }
    }
}

```

Locate Element

```

    }
    if(bToLocateTargetEle){
        if(IS_EQUAL(pRefElement->Ele, targetEle)){
            pElementInfo->pTargetElement = pRefElement;
            pElementInfo->pChainForTargetEle = pRefChain;
            pElementInfo->bTargetEleIsLocated = true;
            bTargetEleIsLocated = true;
        }
    }

    //if bToLocateSourceEle or bToLocateTargetEle is not true and this Element
    has child chain, recursively call to search in the child chain.
    if(pRefElement->pChildChain && (!bSourceEleIsLocated || !bTargetEleIsLocated))
        LocateElement(pRefElement->pChildChain, pElementInfo);
    //if both are found, return, otherwise transit to the next Element.
    if((pElementInfo->bSourceEleIsLocated) && (pElementInfo->bTargetEleIsLocated))
        return;
    }//end of loop for pRefElement
} //end of if(bToLocateSourceEle || bToLocateTargetEle)
else{
    //impossible to be here.
}

```

Fig. 13B

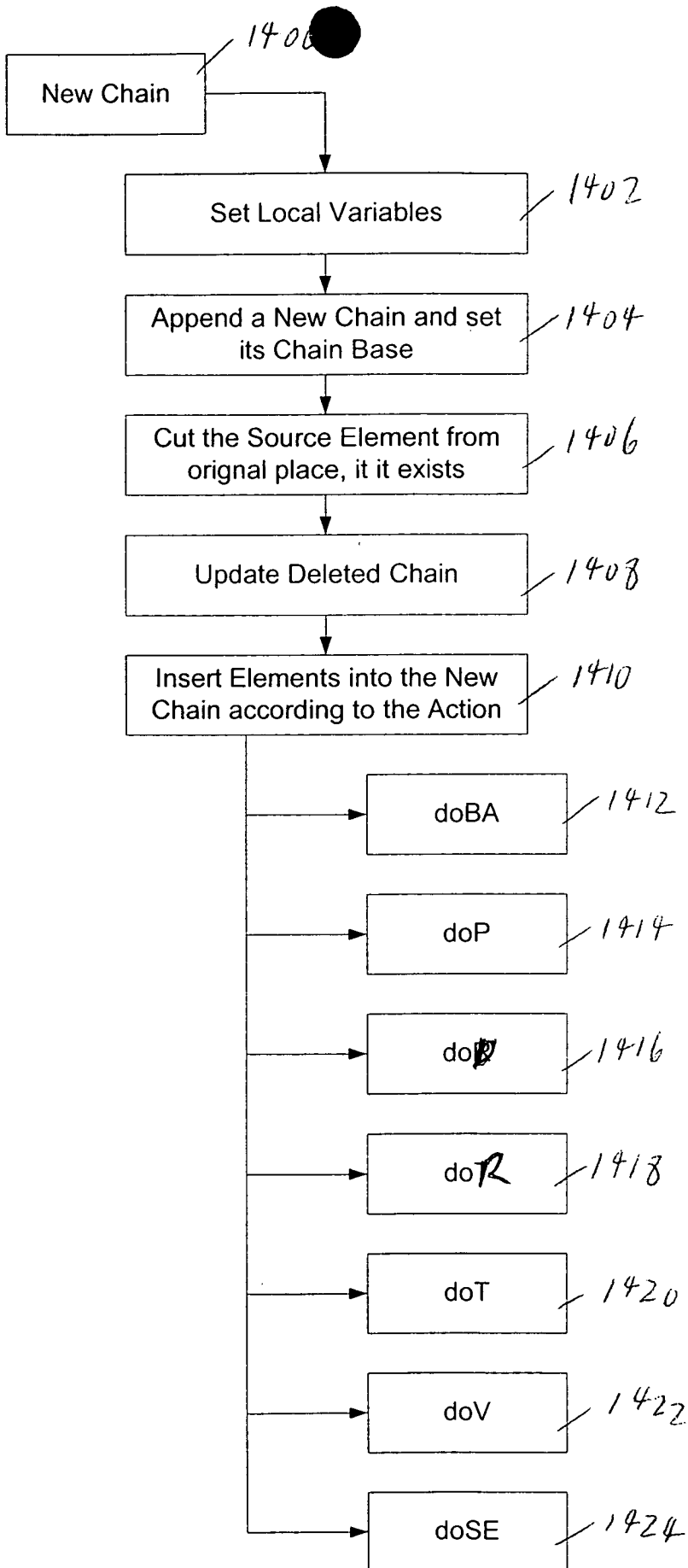


Figure 14A

new chain (1)

Step 1.: Set local variables

```
sourceEle = pStatement->sourceEle; targetEle = pStatement->targetEle;
cAction = pStatement->cAction;
pSourceElement = pElementInfo->pSourceElement;
pTargetElement = pElementInfo->pTargetElement;
Step 2. Append a new chain and set its chain base.
```

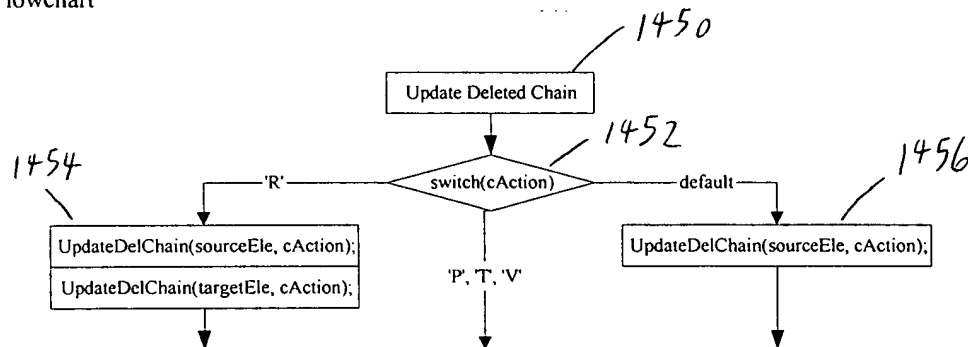
```
if(cAction != 'D'){
    NEW_CHAIN(pChain, targetEle);
    APPEND_CHAIN(pChain);
}
```

Step 3 Cut the source Element from the original place if it exists

```
if(pSourceElement &&
    (cAction == 'B' || cAction == 'A' || cAction == 'S' || cAction == 'E' ||
    cAction == 'R' || cAction == 'D')){
    //Cut the Element from the original place.
    CUT(pElementInfo->pChainForSourceEle->pFirstElement,
        pElementInfo->pChainForSourceEle->pLastElement,
        pSourceElement);
}
//other actions are PTV
```

Step 4 Update deleted chain

Flowchart



Pseudo code

```
switch(cAction){
    case 'R': UpdateDelChain(sourceEle, cAction);
              UpdateDelChain(targetEle, cAction); break;
    case 'P':
    case 'T':
    case 'V': break;
    default: UpdateDelChain(sourceEle, cAction); break; // 'B', 'A', 'S', 'E', 'D'
}
```

new Chain (2)

Step 5 Insert Elements into the new chain according to the action

Top level flowchart for this step

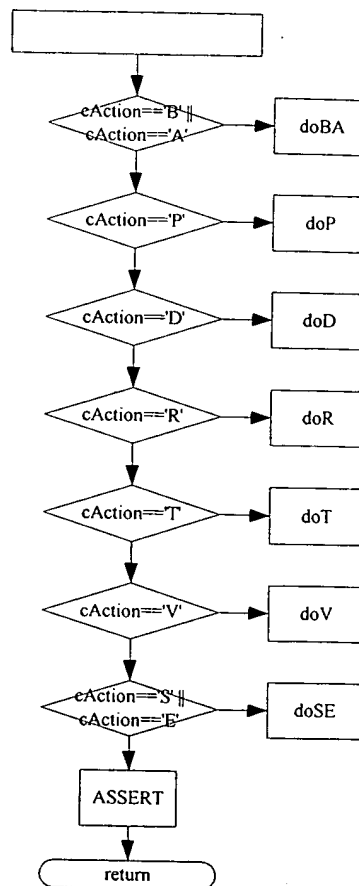


Fig. 14C

Basic Rules for operation newChain

attr: attribute

cA: cAction

SE: sourceEle

TE: targetEle

EE: emptyEle

AP: absolute position

RP: relative position

X(AP): X(sourceEle or targetEle or 'div') with absolute position

X(RP): X(sourceEle or targetEle or 'div') with relative position

XAP: if X(sourceEle or targetEle) has absolute position

XRP: if X(sourceEle or targetEle) has relative position

→X: insert *Element*(X) into the current chain

→X→Y: insert *Element*(X) and *Element*(Y) into the current chain orderly

AP(X): X's absolute position attribute

N_C: NEW_CHAIN(), append the new chain and set its chainBase.

'X': if cAction is 'X'

YES(X), NOT(X): X is true, false;

X(Y): change X's attribute Y.

X(AP)=X(RP)+AP(Y): add Y's absolute position attribute to X.

X(AP)=X(RP)+AP; add user action's AP to X

X(RP)=X(AP)-AP: delete X's AP attribute, then X(AP) becomes X(RP)

→X[→Y→Z]: insert an *Element*(X), into which two children: *Element*(Y) and *Element*(Z) are inserted.

→TE[cA, →SE]: when cA equals 'S', this abbreviation equals →TE[→SE→EE], when cA equals 'E', this abbreviation equals →TE[→EE→SE]

//X: X is comment

SE's position	TE's position	Final position of <i>Element</i> (targetEle) or <i>Element</i> (div)
RP	RP	RP
AP	RP	RP
RP	AP	AP
AP	AP	targetEle's AP

Fig. 14E

3. Examples

After new a chain and set its chain base, insert Elements into the new chain according to the following table.

In this table, "content" means the attributes and text.

In the 2nd column of this table, "▼" means the row involves absolute position.

Action	What user wants to do	What Rule Generator do
B	Insert SE(RP) before TE(RP)	$\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$
	▼ Insert SE(RP) before TE(AP)	AP(div)=AP(TE); TE(RP)=TE(AP)-AP; $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}]$
	▼ Insert SE(AP) before TE(RP)	SE(RP)=SE(AP)-AP; //then SE(AP) becomes SE(RP) $\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$
	▼ Insert SE(AP) before TE(AP)	AP(div)=AP(TE); SE(RP)=SE(AP)-AP; TE(RP)=TE(AP)-AP; $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}]$ //use a div to wrap SE and TE
A	simliar as B	
P	▼ move SE(RP) to AP	SE(AP)=SE(RP)+AP; $\rightarrow \text{SE(AP)}$;
	▼ move SE(AP) to another AP	SE(AP); $\rightarrow \text{SE(AP)}$;
D	Delete this element	
R	Use SE(RP) to replace TE(RP)	$\rightarrow \text{SE(RP)}$;
	▼ Use SE(RP) to replace TE(AP)	SE(AP)=SE(RP)+AP(TE); //SE(RP) becomes SE(AP) $\rightarrow \text{SE(AP)}$
	▼ Use SE(AP) to replace TE(RP)	SE(RP)=SE(AP)-AP; //SE(AP) becomes SE(RP) $\rightarrow \text{SE(RP)}$
	▼ Use SE(AP) to replace TE(AP)	AP(SE)=AP(TE) $\rightarrow \text{SE(AP)}$
T	Change SE' attr	doD; //nothing to do with AP and RP
V	Replace SE's text	doV; //nothing to do with AP and RP
S	Insert SE(RP) to be child of TE(RP)	$\rightarrow \text{TE(RP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(RP) to be child of TE(AP)	$\rightarrow \text{TE(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(AP) to be child of TE(RP)	SE(RP)=SE(AP)-AP; $\rightarrow \text{TE(RP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(AP) to be child of TE(AP)	SE(RP)=SE(AP)-AP; $\rightarrow \text{TE(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
E	similar as S	

Fig. 14F

doBA (1)

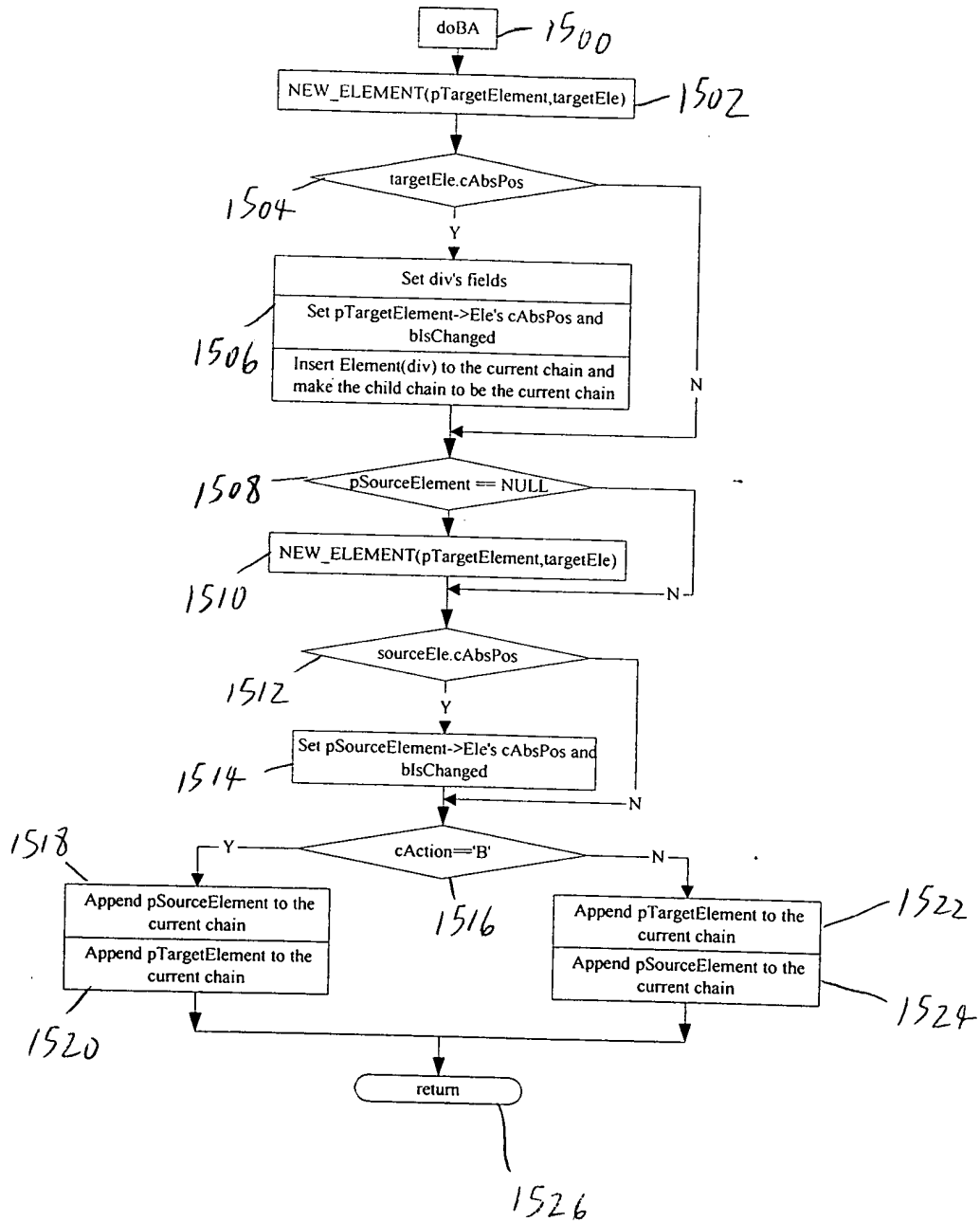


Fig. 15A

Pseudo code

```
//if targetEle has absolute position, a div will be used to wrap sourceEle and targetEle
//The caller of Rule Generator sets the field cAbsPos to indicate the position status of the element:
absolute position or relative position
NEW_ELEMENT(pTargetElement, targetEle);

if(targetEle.cAbsPos){
    //set tag name
    div.sNewTag = "div";
    //Because targetEle does not exist in any chain, the reference absolute position is used for div.
    div.cAbsPos = REF_ABS_POS;

    div.sAbsPos = targetEle.sPath;

    //Set div's sFrame and sPath
    div.sFrame = targetEle.sFrame;
    div.sPath = targetEle.sPath;

    //set blsAbsPosOrg and cAbsPos to indicate that the absolute position attribute shall not be
    output when XSLT is applied.
    pTargetElement->Ele.cAbsPos = NO_ABS_POS;

    //set blsChanged to indicate targetEle is changed.
    pTargetElement->Ele.blsChanged = true;

    //new an Element for div and append to the current new chain
    NEW_ELEMENT(pDivElement, div);
    pDivElement->blsChainBase = true;
    APPEND_ELEMENT(pChain, pDivElement);

    //make a new chain to be the child chain of div
    pChain = new Chain;
    pDivElement->pChildChain = pChain;
}
else ///!
    pTargetElement->blsChainBase = true;

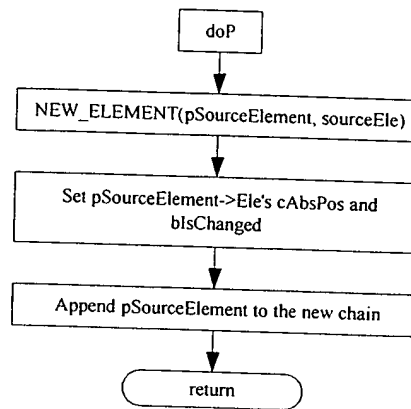
//if sourceEle does not exist in any chain, make a new Element for sourceEle
if( ! pSourceElement){
    NEW_ELEMENT(pSourceElement, sourceEle);
}
```

```
//if sourceEle is absolute position, set blsAbsPosOrg, cAbsPos and blsChanged to indicate that the
absolute position attribute of sourceEle shall not be output and sourceEle is changed.
if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}

//append pSourceElement and pTargetElement according to the action
if(cAction == 'B'){
    APPEND_ELEMENT(pChain, pSourceElement);
    APPEND_ELEMENT(pChain, pTargetElement);
}
else{
    APPEND_ELEMENT(pChain, pTargetElement);

    APPEND_ELEMENT(pChain, pSourceElement);
}
}
```

doP



Pseudo code

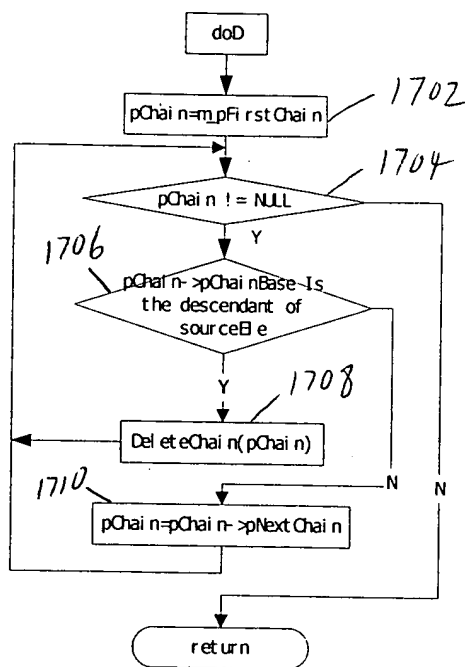
When the statement is pushed into the redoStack, if the action is PTV, sourceEle and targetEle is set to the same value. So when targetEle is not found in any chain, the sourceEle shall not appear in any chain. Therefore a new *Element* is made for sourceEle.

```
NEW_ELEMENT(pSourceElement, sourceEle);
pSourceElement->Ele.cAbsPos = REAL_ABS_POS;
pSourceElement->Ele.bIsChanged = true;
//the position (x, y) has been set by the caller of Rule Generator.
```

```
pSourceElement->bIsChainBase = true;
APPEND_ELEMENT(pChain, pSourceElement);
```

Fig. 16

200



Pseudo Code

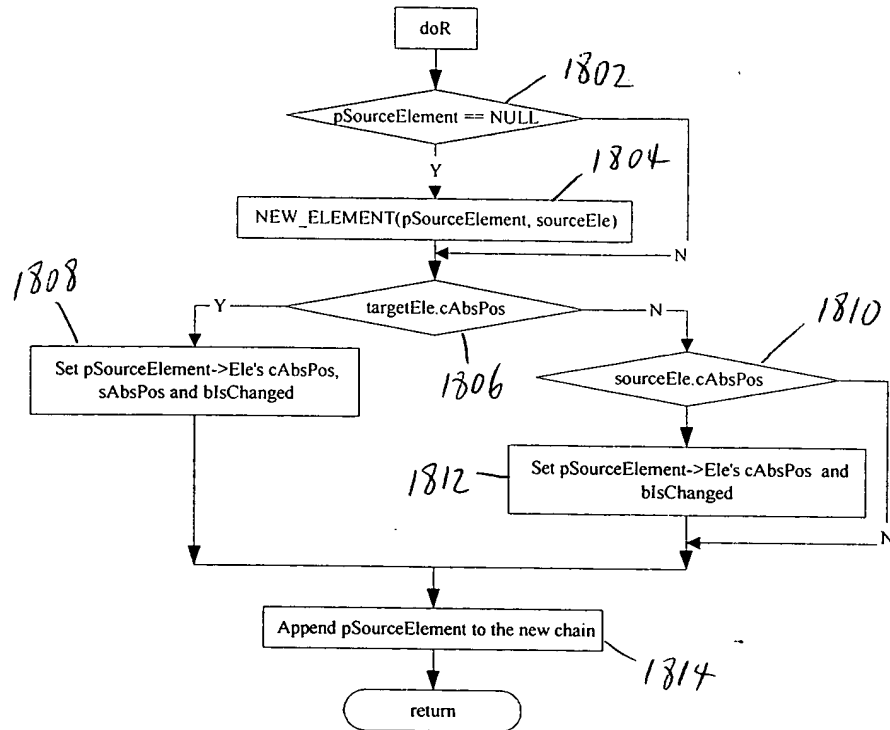
Check all chains. If the chain base of a chain is the descendant of sourceEle, delete that chain.

```

for(pChain = m_pFirstChain; pChain != NULL;){
    if(IS_DESCENDANT (pChain->pChainBase->Ele, sourceEle))
        pChain = DeleteChain(pChain); //return the pointer of the next chain
    else
        pChain = pChain->pNext;
}
  
```

Fig. 17

doR



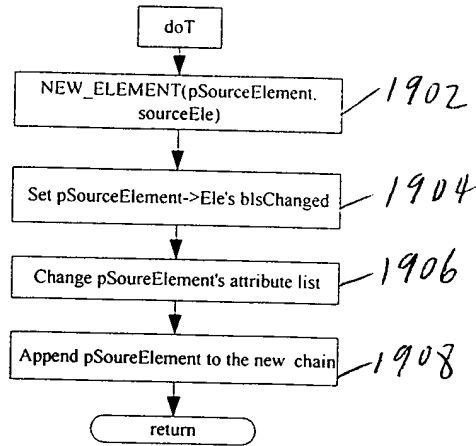
Pseudo code

```

if( ! pSourceElement ){
    NEW_ELEMENT(pSourceElement, sourceEle);
}
if(targetEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = REF_ABS_POS;
    pSourceElement->Ele.sAbsPos = targetEle.sPath;
    pSourceElement->Ele.bIsChanged = true;
}
else{
    if(sourceEle.cAbsPos){
        pSourceElement->Ele.cAbsPos = NO_ABS_POS;
        pSourceElement->Ele.bIsChanged = true;
    }
}
APPEND_ELEMENT(pChain, pSourceElement);
  
```

Fig. 18

doT



Pseudo code

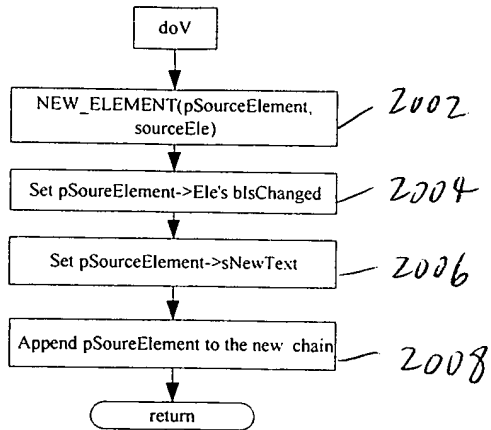
```
//set the Element
NEW_ELEMENT(pSourceElement, sourceEle);
pSourceElement->Ele.blsChanged = true;

//scan the attribute list

psNewAttrName = pStatement->psNewAttrName;
psNewAttrValue = pStatement->psNewAttrValue;
for(i=0; i<pStatement->iNumOfAttr; i++){
    pAttr = new Attr;
    //set the body of pAttr
    pAttr->sAttrName = psNewAttrName[i];
    pAttr->sAttrValue = psNewAttrValue[i];
    pAttr->pPrev = pAttr->pNext = NULL;
    APPEND_ATTR(pSourceElement, pAttr);
}
//append the source Element to the chain
pSourceElement->blsChainBase = true;
APPEND_ELEMENT(pChain, pSourceElement);
```

Fig. 19

doV

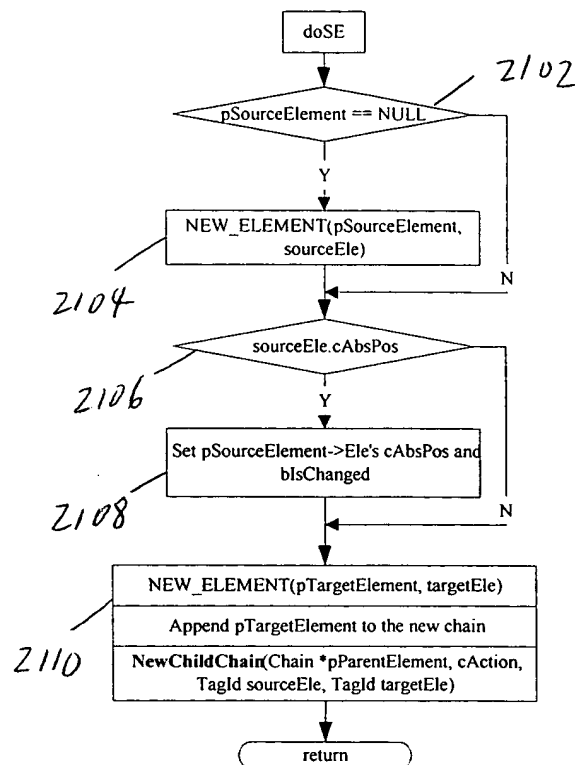


Pseudo code

```
NEW_ELEMENT(pSourceElement, sourceEle);  
pSourceElement->Ele.bIsChanged = true;  
  
pSourceElement->sNewText = pStatement->sNewText;  
  
pSourceElement->bIsChainBase = true;  
APPEND_ELEMENT(pChain, pSourceElement);
```

Fig. 20

doSE



Pseudo Code

```

pTargetElement->Ele.blsChanged = true;

if (! pSourceElement){
    NEW_ELEMENT(pSourceElement, sourceEle);
}

if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}

NEW_ELEMENT(pTargetElement, targetEle);

pTargetElement->bIsChainBase = true;
APPEND_ELEMENT (pChain, pTargetElement);
NewChildChain(pTargetElement, cAction, pSourceElement);
  
```

Fig. 21

new child chain

Pseudo code

//New a child chain for the input Element

NEW_CHILD_CHAIN(pChain);

pParentElement->pChildChain = pChain;

pChain->pParentElement = pParentElement;

//New a empty *Element*

emptyEle.iFamilyId = -1; // -1 represent this is a empty *Element*

NEW_ELEMENT(pEmptyElement, emptyEle);

//Append two Elements according to the action

if(cAction == 'S'){

 APPEND_ELEMENT(pChain, pSourceElement);

 APPEND_ELEMENT(pChain, pEmptyElement);

}

else{//cActio == 'E'

 APPEND_ELEMENT(pChain, pEmptyElement);

 APPEND_ELEMENT(pChain, pSourceElement);

}

Fig. 22

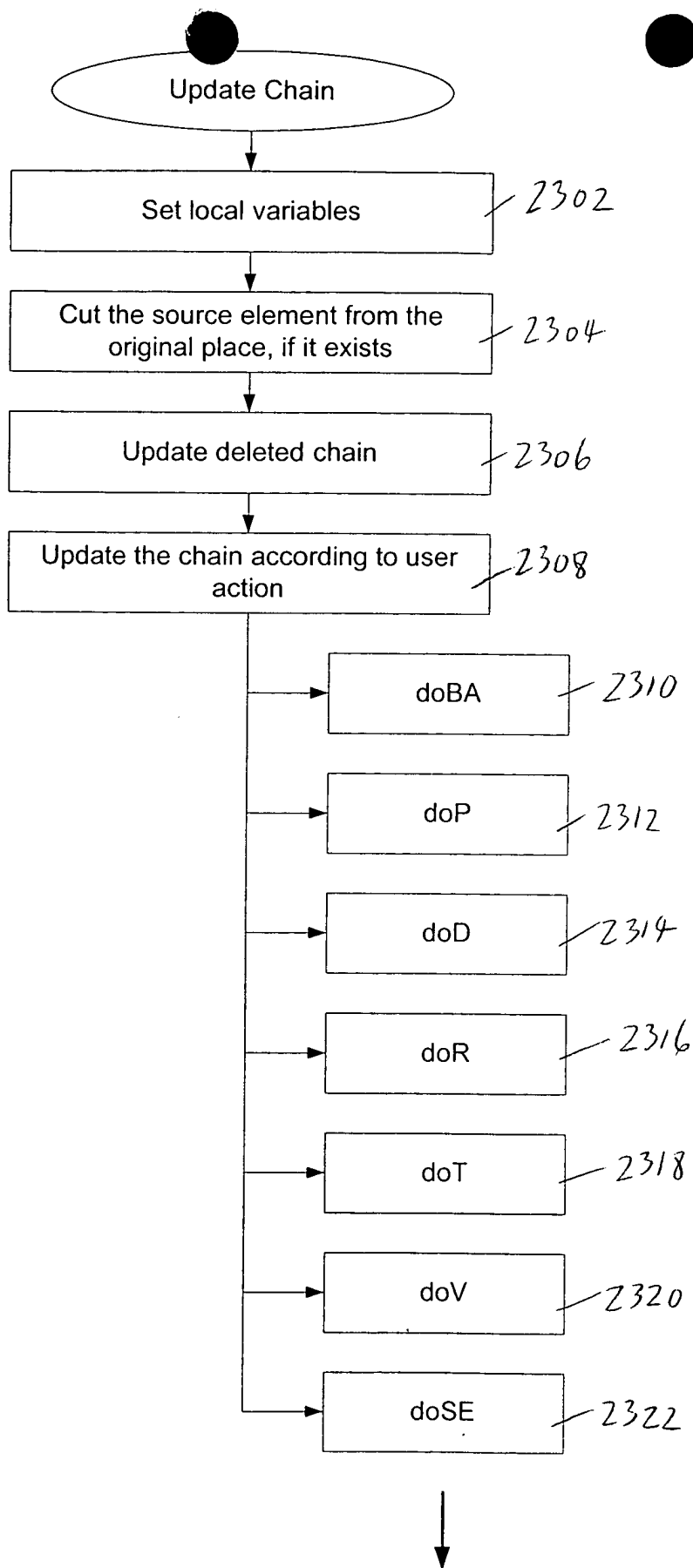


Figure 23.4

Update Chain (1)

Step 1 Set local variables

```
sourceEle = pStatement->sourceEle; targetEle = pStatement->targetEle;
cAction = pStatement->cAction;
pSourceElement = pElementInfo->pSourceElement;
pTargetElement = pElementInfo->pTargetElement;

pChainForSourceEle = pElementInfo->pChainForSourceEle;
pChainForTargetEle = pElementInfo->pChainForTargetEle;
```

Step 2 Cut the source Element from the original place if it exists(the same as that in NewChain)

```
if(pSourceElement &&
    (cAction == 'B' || cAction == 'A' || cAction == 'S' || cAction == 'E' ||
     cAction == 'R' || cAction == 'D'))
{
```

//Cut the Element from the original place.

```
CUT(pElementInfo->pChainForSourceEle->pFirstElement,
    pElementInfo->pChainForSourceEle->pLastElement,
    pSourceElement);
```

```
}
```

//other actions are PTV

Step 3 Update Deleted Chain (the same as that in NewChain)

```
switch(cAction){
    case 'R': UpdateDelChain(sourceEle, cAction);
              UpdateDelChain(targetEle, cAction);break;
    case 'T':
    case 'V': break;
    default: UpdateDelChain(sourceEle, cAction); break;// 'B' 'A' 'S', 'E', 'D'
}
```

Update Chain C

Step 4 Update the chain according to user action

Top level flowchart

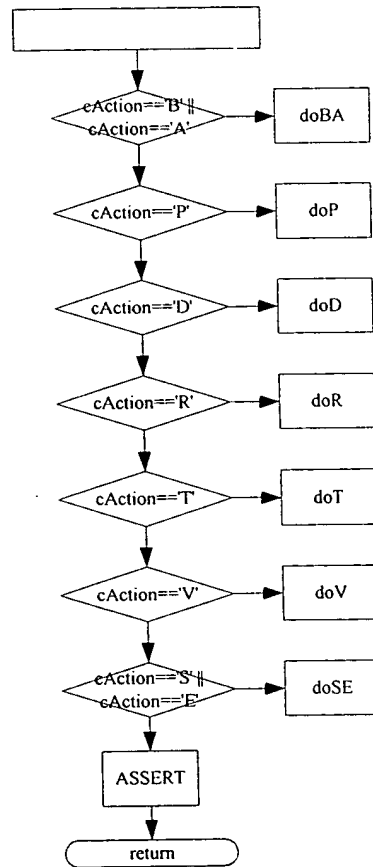


Fig. 23C

Basic rules for the above operation *Update Chain*

1. Abbreviation

Two abbreviations are defined besides the ones in NewChain

$\rightarrow \{X\}$: X is an Element that is already in the current chain.

$\rightarrow X \rightarrow \forall$: Replace *Element*(Y) with *Element*(X).

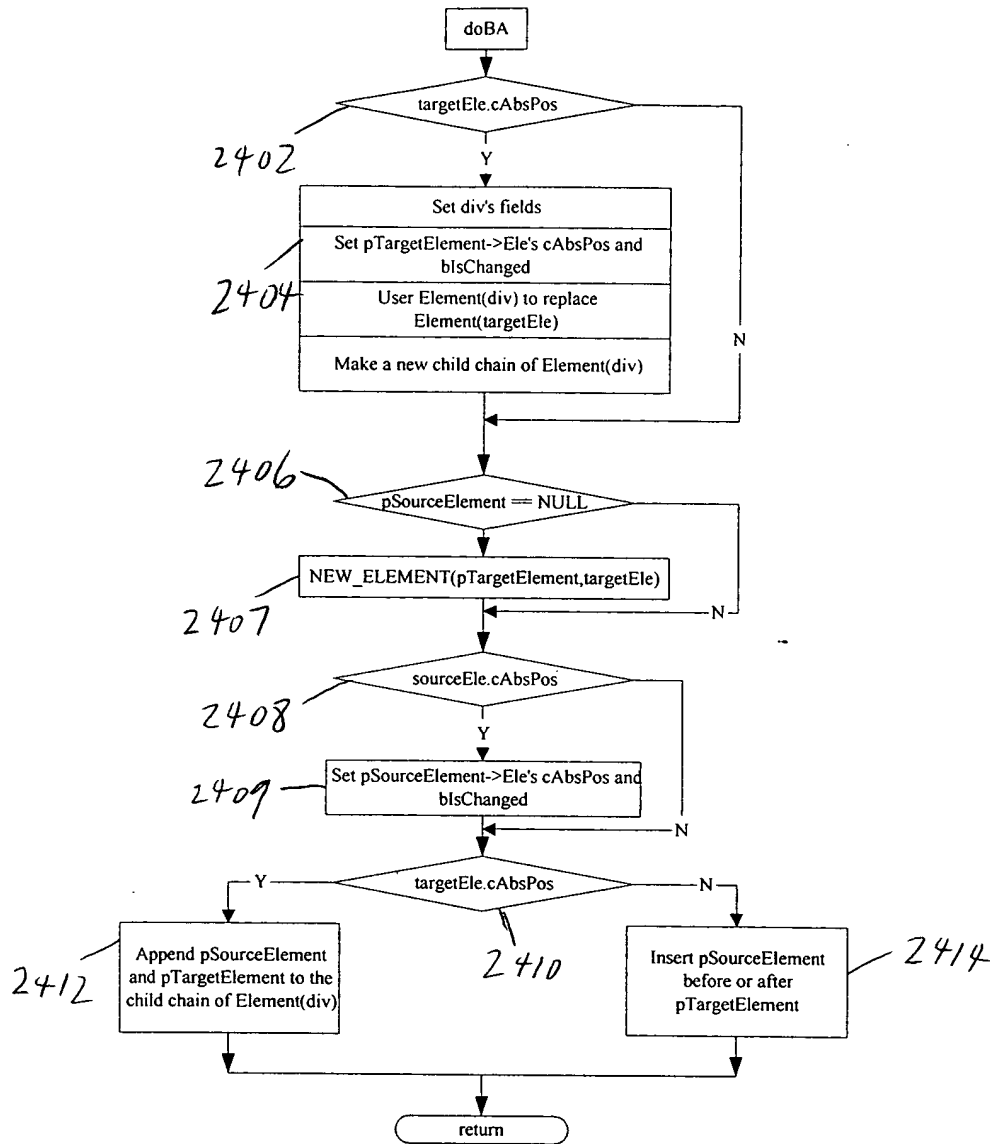
2. Handle different positions(the same as that in NewChain)

3. Examples

Action	What user wants to do	What Rule Generator do
B	Insert SE(RP) before TE(RP)	$\rightarrow \text{SE(RP)} \rightarrow \{\text{TE(RP)}\}$
	▼ Insert SE(RP) before TE(AP)	AP(div)=AP(TE); TE(RP)=TE(AP)-AP; $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}] \rightarrow \text{TE(AP)}$
	▼ Insert SE(AP) before TE(RP)	SE(RP)=SE(AP)-AP; //then SE(AP) becomes SE(RP) $\rightarrow \text{SE(RP)} \rightarrow \{\text{TE(RP)}\}$
	▼ Insert SE(AP) before TE(AP)	AP(div)=AP(TE); SE(RP)=SE(AP)-AP; TE(RP)=TE(AP)-AP; $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}] \rightarrow \text{TE(AP)}$ //use a div to wrap SE and TE
A	similiar as B	
P	▼ move SE(RP) to AP	SE(AP)=SE(RP)+AP;
	▼ move SE(AP) to another AP	SE(AP);
D	Delete this element	
R	Use SE(RP) to replace TE(RP)	$\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$;
	▼ Use SE(RP) to replace TE(AP)	SE(AP)=SE(RP)+AP(TE); //SE(RP) becomes SE(AP) $\rightarrow \text{SE(AP)} \rightarrow \text{TE(AP)}$
	▼ Use SE(AP) to replace TE(RP)	SE(RP)=SE(AP)-AP; //SE(AP) becomes SE(RP) $\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$
	▼ Use SE(AP) to replace TE(AP)	AP(SE)=AP(TE) $\rightarrow \text{SE(AP)} \rightarrow \text{TE(AP)}$
T	Change SE' attr	doD; //nothing to do with AP and RP
V	Replace SE's text	doV; //nothing to do with AP and RP
S	Insert SE(RP) to be child of TE(RP)	$\rightarrow \{\text{TE(RP)}\}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(RP) to be child of TE(AP)	$\rightarrow \{\text{TE(AP)}\}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(AP) to be child of TE(RP)	SE(RP)=SE(AP)-AP; $\rightarrow \{\text{TE(RP)}\}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	▼ Insert SE(AP) to be child of TE(AP)	SE(RP)=SE(AP)-AP; $\rightarrow \{\text{TE(AP)}\}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
E	similar as S	

Fig. 230

doBA (1)



Pseudo code

```

//if targetEle has absolute position, a div will be used to wrap sourceEle and targetEle
//The caller of Rule Generator sets the field cAbsPos to indicate the position status of the element:
absolute position or relative position
if(targetEle.cAbsPos){
    //set tag name
    div.sNewTag = "div";
    //set targetEle's absolute position attribute to div
    div.cAbsPos = targetEle.cAbsPos;
    //when cAbsPos is REF_ABS_POS, sPath is used, otherwise (x, y) is used.
    div.sAbsPos = targetEle.sAbsPos;

```

Fig. 24A

40A

DOBA(2)

```

div.x = targetE
div.y = targetEle.y;
//set targetEle.cAbsPos to NO_ABS_POS indicate that the absolute position attribute shall
not be output when XSLT is applied.
pTargetElement->Ele.cAbsPos = NO_ABS_POS;

//set blsChanged to indicate targetEle is changed.
pTargetElement->Ele.blsChanged = true;

//new an Element for div and append to the current new chain
NEW_ELEMENT(pDivElement, div);
//use Element(div) to replace Element(targetEle)
REPLACE(pChainForTargetEle->pFirstElement, pChainForTargetEle->pLastElement,
pTargetElement, pDivElement);

//make a new chain to be the child chain of div
pDivChain = new Chain;
pDivElement->pChildChain = pDivChain;
}

//if sourceEle does not exist in any chain, make a new Element for sourceEle
if (! pSourceElement){
    NEW_ELEMENT(pSourceElement, sourceEle);
}

//if sourceEle is absolute position, set blsAbsPosOrg, cAbsPos and blsChanged to indicate that the
absolute position attribute of sourceEle shall not be output and sourceEle is changed.
if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}

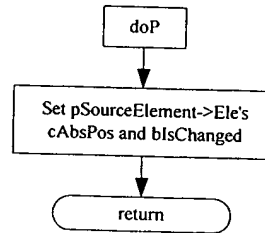
if(targetEle.cAbsPos){
    if(cAction == 'B'){
        APPEND_ELEMENT (pDivChain, pSourceElement);
        APPEND_ELEMENT (pDivChain, pTargetElement);
    }
    else{
        APPEND_ELEMENT (pDivChain, pTargetElement);
        APPEND_ELEMENT (pDivChain, pSourceElement);
    }
}
else{
    if(cAction == 'B'){
        INSERT_BEFORE(pChainForTargetEle->pFirstElement, pSourceElement, pTargetElement);
    }
    else{
        INSERT_AFTER (pChainForTargetEle->pFirstElement, pTargetElement, pSourceElement);
    }
}
}

```


doP

• doP:

Flowchart

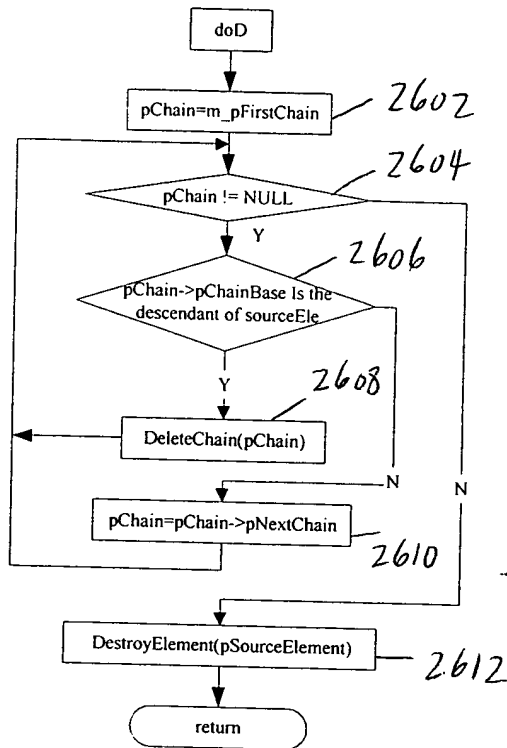


Pseudo code

pSourceElement->Ele.cAbsPos = REAL_ABS_POS;
pSourceElement->Ele.blsChanged = true;

Fig. 25:

doD

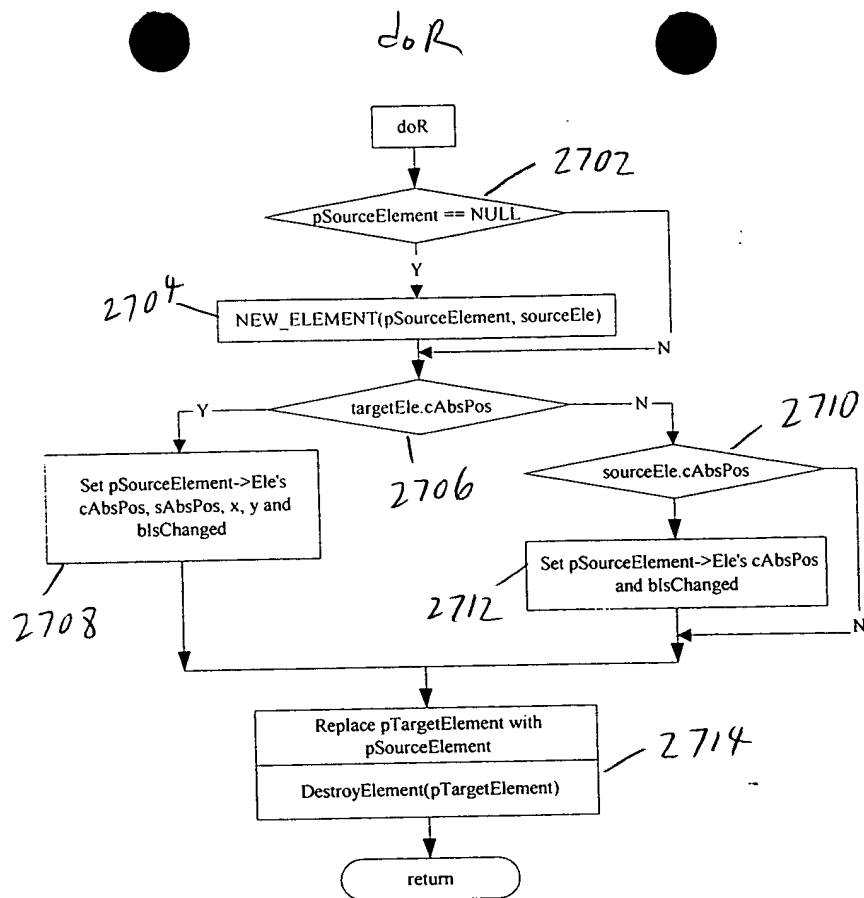


Pseudo code

```

for(pChain = m_pFirstChain; pChain != NULL;){
    if(IS_DESCENDANT(pChain->pChainBase->Ele, pSourceElement->Ele))
        pChain = DeleteChain(pChain); //return the pointer of the next chain
    else
        pChain = pChain->pNext;
}
//pSourceElement has been cut from the original place and it is totally destroyed now.
DestroyElement(pSourceElement);
  
```

Fig. 26



Pseudo code

```
if(! pSourceElement ) NEW_ELEMENT(pSourceElement, sourceEle);
```

```
if(targetEle.cAbsPos){
```

```
    //set targetEle's absolute position attribute to sourceEle
```

```
    pSourceElement->Ele.cAbsPos = targetEle.cAbsPos;
```

```
    pSourceElement->Ele.sAbsPos = targetEle.sAbsPos;
```

```
    pSourceElement->Ele.x = targetEle.x;
```

```
    pSourceElement->Ele.y = targetEle.y;
```

```
    pSourceElement->Ele.bIsChanged = true;
```

```
}
```

```
else{
```

```
    if(sourceEle.cAbsPos){
```

```
        pSourceElement->Ele.cAbsPos = NO_ABS_POS;
```

```
        pSourceElement->Ele.bIsChanged = true;
```

```
    }
```

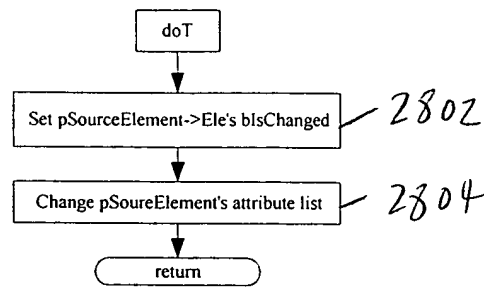
```
}
```

```
REPLACE(pChainForTargetEle->pFirstElement,  
pTargetElement, pSourceElement)
```

```
pChainForTargetEle->pLastElement,
```

```
//after replacement, pTargetElement is cut from the original place, so destroy it now.
```

doT



Pseudo code

```

pSourceElement->Ele.blIsChanged = true;

//scan the attribute list
psNewAttrName = pStatement->psNewAttrName;
psNewAttrValue = pStatement->psNewAttrValue;
for(i=0; i<pStatement->iNumOfAttr; i++){
    for(pAttr=pSourceElement->pFirstAttr; pAttr != NULL; pAttr=pAttr->pNext){
        if(psNewAttrName[i] == pAttr->sAttrName){
            pAttr->sAttrValue = psNewAttrValue[i];
            if(pAttr->sAttrValue.length() == 0){
                CUT(pSourceElement->pFirstAttr, pSourceElement->pLastAttr, pAttr);
                pTempAttr = pSourceElement->pFirstAttr;
                INSERT_BEFORE(pSourceElement->pFirstAttr, pAttr, pTemp);
            }
            break;//jump to where
        }
    }
    //not found
    pAttr = new Attr;
    //set the body of pAttr
    pAttr->sAttrName = psNewAttrName[i];
    pAttr->sAttrValue = psNewAttrValue[i];
    pAttr->pPrev = pAttr->pNext = NULL;

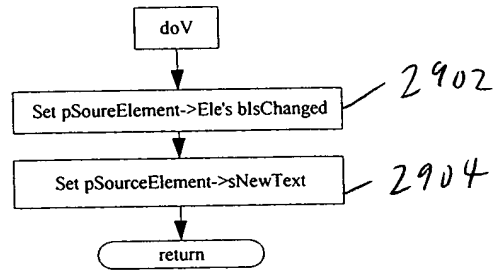
    if(pAttr->sAttrValue.length()){
        APPEND_ATTR(pSourceElement, pAttr);
    }
    else{
        pTempAttr = pSourceElement->pFirstAttr;
        INSERT_BEFORE(pSourceElement->pFirstAttr, pAttr, pTempAttr);
    }
}
  
```

Fig. 28

doV

• doV:

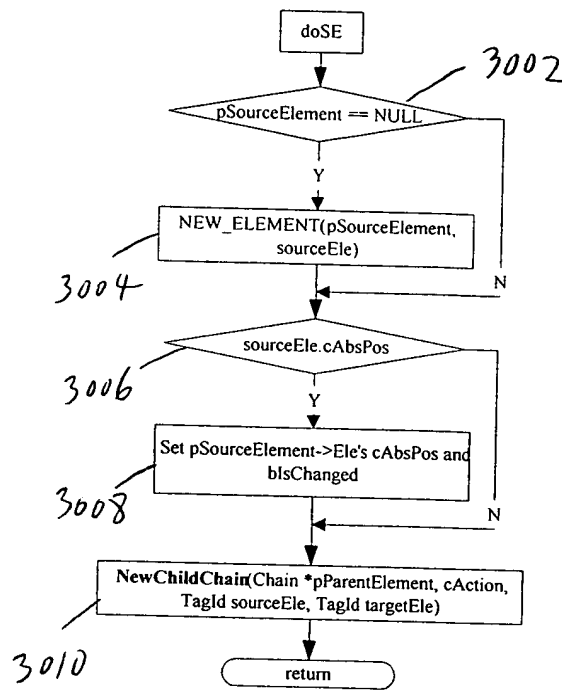
Flowchart



Pseudo code

```
pSourceElement->Ele. blsChanged = true;  
pSourceElement->sNewText = pStatement->sNewText;
```

doSE



Pseudo code

```

pTargetElement->Ele.blsChanged = true;

if( ! pSourceElement) NEW_ELEMENT(pSourceElement, sourceEle);
if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}

if(pTargetElement->pChildChain){
    if(cAction == 'S'){
        pElement = pTargetElement->pChildChain->pFirstElement;
        INSERT_BEFORE(pTargetElement->pChildChain->pFirstElement, pSourceElement,
        pElement);
    }
    else{
        APPEND_ELEMENT(pTargetElement->pChildChain, pSourceElement);
    }
}
else
    NewChildChain(pTargetElement, cAction, pSourceElement);
  
```

Fig. 30

Update del chain

```
pRefElement = m_pDelChain->pFirstElement;
while(pRefElement != NULL){
    if(IS_EQUAL(pRefElement->Ele, Ele)){
        pRefElement->cAction = cAction;
        return;
    }
    pRefElement = pRefElement ->pNext;
}

NEW_ELEMENT(pElement, Ele);
APPEND_ELEMENT(m_pDelChain, pElement);
return;
```

Fig. 31

```

pCurrElement = m_DelChain->pFirstElement;
while(pCurrElement != NULL){
    if(Element is from source file){
        //look for its yougest ancestor
        bHasAncestor = false;
        pRefElement = m_pDelChain->pFirstElement;
        for( ; pRefElement != NULL; pRefElement = pRefElement->pNext) {
            if(pRefElement != pCurrElement){
                if(IS_DESCENDANT(pRefElement->Ele, pCurrElement->Ele)){
                    if(bHasAncestor){
                        if(IS_DESCENDANT(pYougestAncestor->Ele, pRefElement->Ele)){
                            pYougestAncestor = pRefElement;
                        }
                    }
                }
            }
        }
        else{
            bHasAncestor = true;
            pYougestAncestor = pRefElement;
        }
    }
    if(bHasAncestor)
        if(pYougestAncestor->cAction != 'D')
            //Note: when put an exist Element into Deleted Chain, that action of Element is overwritten.
            pCurrElement->Ele.cAbsPos = false;//mark it not to be deleted
        else
            pCurrElement->Ele.cAbsPos = true;
    else
        pCurrElement->Ele.cAbsPos = true;//has no ancestor, mark it to be deleted later
    }//if(element is in source file
    else
        pCurrElement->Ele.cAbsPos = false;
    pCurrElement = pCurrElement->pNext;
}

//Remove Elements whose cAbsPos = true
pCurrElement=m_pDelChain->pFirstElement;
while(pCurrElement != NULL){
    if(pCurrElement->Ele.cAbsPos)
        pCurrElement=DeleteElement(&m_DelChain, pCurrElement);
    else
        pCurrElement=pCurrElement->pNext;
}

```


Assembly Chain

```

//Assembly chains cards and pages according to the field iPage and iCard of structure TagId
for( pChain = m_pFirstChain; pChain != NULL; pChain = pChain->pNext){
    for(pPage = m_pFirstPage; pPage != NULL; pPage = pPage->pNext){
        if(pChain->pChainBase->Ele.iPage == pPage->iPage){
            for( pCard = pPage->pFirstCard; pCard != NULL; pCard = pCard->pNext){
                if(pChain->pChainBase->Ele.iCard == pCard->iCard){
                    CUT(m_pFirstChain, m_pLastChain, pChain);
                    APPEND_CHAIN_TO_CARD(pCard, pChain);

                    if(pCard->iEntry == HAS_NOT_ENTRY)
                        pCard->iEntry = pChain->pChainBase->Ele.iFamilyId;
                    goto NEXT_CHAIN;
                }
            }
        }
    }
}

NEW_PAGE(pPage, pChain->pChainBase->Ele.iPage)
APPEND_PAGE(this, pPage);
NEW_CARD(pCard, pChain->pChainBase->Ele.iCard);
APPEND_CARD(pPage, pCard);
CUT(m_pFirstChain, m_pLastChain, pChain);
APPEND_CHAIN_TO_CARD(pCard, pChain);

if(pCard->iEntry == HAS_NOT_ENTRY){
    pCard->iEntry = pChain->pChainBase->Ele.iFamilyId;
NEXT_CHAIN:
;
}

//Assembly Elements in Deleted Chains according to FamilyId
for(pElement = m_pDelChain->pFirstElement; pElement != NULL; pElement =
pElement->pNext){
    for(pChain = m_pFirstDelChain; pChain != NULL; pChain = pChain->pNext){
        if(pElement->Ele.iFamilyId == pChain->pFirstElement->Ele.iFamilyId){
            CUT(m_pDelChain->pFirstElement, m_pDelChain->pLastElement, pElement);
            APPEND_ELEMENT(pChain, pElement);
            goto NEXT_DEL_ELEMENT;
        }
    }
}

NEW_CHILD_CHAIN(pChain);
APPEND_DEL_CHAIN(pChain);
APPEND_ELEMENT(pChain, pElement);
NEXT_DEL_ELEMENT:
;

```

parseframe (1)

```
//
if(pChain->pChainBase)
    pElement = pChain->pChainBase;
else
    pElement = pChain->pFirstElement;

bTmpHasFrame = bSrcHasFrame = true;

while(pElement != NULL && (bSrcHasFrame || bTmpHasFrame)){

    if(pElement->bIsChainBase) bChainBaseIsFound = true;

    if(pElement->Ele.sPath.charAt(0) == 't'){
        cFrom = TEMPLATE;
        if(bTmpHasFrame)
            pVar = pPage->pRootTmpVar;
        else
            goto NEXT_ELEMENT;
    }
    else if(pElement->Ele.sPath.charAt(0) == 's'){
        cFrom = SOURCE;
        if(bSrcHasFrame)
            pVar = pPage->pRootSrcVar;
        else
            goto NEXT_ELEMENT;
    }
    else{
        //DEBUG
        printf("wrong path!\n");
    }

    sFrame = pElement->Ele.sFrame;
    iMaxLayer = GetLayerNumber(sFrame);
}
```

Fig. 34A

50A

```

for(i=0; i<iMaxLayer; i++){

    iFirstDotPos = GetFirstCharPos(sFrame, '.');

    if(iFirstDotPos != -1){
        sTemp = sFrame.substringData(0, iFirstDotPos);
        sFrame = sFrame.substringData(iFirstDotPos+1, sFrame.length()-iFirstDotPos);
    }
    else
        sTemp = sFrame;

    if(sTemp.charAt(0) == 'I'){
        bIsFrame = false;
        sTemp = sTemp.substringData(1, sTemp.length()-1);
    }
    else
        bIsFrame = true;

    if(sTemp.length() == 0){ //the page containing this element has no frame.

        if(cFrom == TEMPLATE)
            bTmpHasFrame = false;
        else
            bSrcHasFrame = false;

        goto NEXT_ELEMENT;
    }
    else
        iFrameIndex = atoi(sTemp.transcode());

    if(bIsFrame){
        if(iFrameIndex < pVar->iMaxFrame){
            pVar = pVar->pFirstFrame;
            for(i=1; i<iFrameIndex; i++){
                pVar = pVar->pNext;
            }
            goto NEXT_ELEMENT;
        }
        else{
            for(j = pVar->iMaxFrame; j < iFrameIndex; j++){
                itoa(j+1, szBuffer, 10);
                sCurrFrame = szBuffer;
            }
        }
    }
}

```

```

        if(pVar != pPage->pRootSrcVar && pVar != pPage->pRootTmpVar)
            sCurrFrame = pVar->sFrame + "." + sCurrFrame;

        NEW_VAR(pNewVar, sCurrFrame);
        APPEND(pVar, pNewVar, Frame);
    } //loop for j
    pVar->iMaxFrame = iFrameIndex;
    pVar = pVar->pLastFrame;
}
}
else{
    if(iFrameIndex < pVar->iMaxIFrame){
        pVar = pVar->pFirstIFrame;
        for(i=1; i<iFrameIndex; i++){
            pVar = pVar->pNext;
        }
        goto NEXT_ELEMENT;
    }
    else{
        for(j = pVar->iMaxIFrame; j < iFrameIndex; j++){
            itoa(j+1, szBuffer, 10);

            sCurrIFrame = szBuffer;

            if(pVar != pPage->pRootSrcVar && pVar != pPage->pRootTmpVar)
                sCurrFrame = pVar->sFrame + "." + sCurrFrame;

            NEW_VAR(pNewVar, sCurrIFrame);
            APPEND(pVar, pNewVar, IFrame);
        }
        pVar->iMaxIFrame = iFrameIndex;
        pVar = pVar->pFirstIFrame;
    }
}

pVar->bToOutput = true;
}

NEXT_ELEMENT:

    if(pChain->pChainBase == pElement)
        pElement = pChain->pFirstElement;
    else
        pElement = pElement->pNext;
} //end of loop for pElement

```

```

if(pVar->bToOutput == false) return;

if(pVar->sFrame contains the character '.'){
    iLastDotPos = pVar->sFrame.lastIndexOf(pVar->sFrame, '.');
    iLength = pVar->sFrame.length();

    sParentFrame = pVar->sFrame.substringData(0, iLastDotPos);
    sSelfFrame = pVar->sFrame.substringData(iLastDotPos + 1, iLength - iLastDotPos - 1);
}
else{
    sParentFrame = "";

    sSelfFrame = pVar->sFrame;
}

if(the first character of sSelfFrame is 'i'){
    //it is a "iframe"
    bIsFrame = true;
    sSelfFrame = pVar->sFrame.substringData(iLastDotPos + 1, iLength - iLastDotPos - 1);
}

//Set frame type
sFrameType = bIsFrame ? "iframe" : "frame"

//Output the variable for this frame
<<xml:variable name="sFrom" pVar->sFrame" select="document($
    sFrom sParentFrame//sFrameType [ sSelfFrame ]/@src")/>

//Iteratively output the XSLT variables for the frames of the current var
for(pFrame = pVar->pFirstFrame; pFrame ; pFrame = pFrame->pNext){
    OutputVar(xsltFile, pFrame, sFrom);
}

//Iteratively output the XSLT variables for the iframes of the current var
for(pIFrame = pVar->pFirstIFrame; pIFrame ; pIFrame = pIFrame->pNext){
    OutputVar(xsltFile, pIFrame, sFrom);
}

```

Fig. 35

```
pElement = pChain->pFirstElement;
```

```
bChainBaseIsFound = false;
```

```
while(pElement){
```

```
    if(pElement->bIsChainBase) bChainBaseIsFound = true;
```

```
    if(pElement->Ele.iFamilyId != EMPTY_ELEMENT){
```

```
        szRoot = ((pElement->Ele.sPath.substringData(0, 3))
```

```
            + (pElement->Ele.sFrame)).transcode();
```

```
        <<xsl:apply-templates select="szRoot szPath"
```

```
            mode="szRoot output pElement->Ele.iFamilyId"/>
```

```
        NEW_UNIT(pUnit, pElement);
```

```
        if(pElement->Ele.bIsChanged == true || pElement->Ele.sNewTag.length() != 0)
```

```
            //this Element is changed
```

```
            APPEND_UNIT(pCard, pUnit); //the Queue will be scanned later
```

```
        }
```

```
        else{
```

```
            APPEND_UNIT_TO_FAMILY(pCard, pUnit);
```

```
        }
```

```
    } else { //it is the empty Element which is the content of the parent Element of this chain
```

```
        szRoot = ((pChain->pParentElement->Ele.sPath.substringData(0, 3))
```

```
            + (pChain->pParentElement->Ele.sFrame)).transcode();
```

```
        <<xsl:apply-templates select="*|text()" mode="szRoot test_
```

```
            pChain->pParentElement->Ele.iFamilyId"/>
```

```
    }
```

```
    pElement = pElement->pNext;
```

```
} //end of while
```

```
//if the chain has chain base and the chain base is not in the chain (i.e. it is moved to other chains  
or it is deleted), append the chain base to family according to it family id.
```

```
if(!bChainBaseIsFound && pChain->pChainBase){
```

```
    NEW_UNIT(pUnit, pChain->pChainBase);
```

```
    APPEND_UNIT_TO_FAMILY(pCard, pUnit);
```

```
}
```

Get Unit

```
if(pCard->pFirstUnit){
    pUnit = pCard->pFirstUnit;
    pElement = pCard->pFirstUnit->pElement;
    pCard->pFirstUnit = pCard->pFirstUnit->pNext;
    pCard->pFirstUnit->pPrev = NULL;
    free(pUnit);
    return pElement;
}
else
    return NULL;
```

Fig. 37

2.4. Macros

The following macros are in macro.h

2.4.1. Constant Value

```
#define SOURCE          0
#define TEMPLATE        1

#define NO_ABS_POS      0
#define REF_ABS_POS     1
#define REAL_ABS_POS    2

#define FROM_NOWHERE    0
#define FROM_M_PFIRSTCHAIN 1
#define FROM_CARD       2
#define FROM_M_PFIRSTDELCHAIN 3
```

2.4.2. INIT

2.4.2.1. INIT_STACK

```
#define INIT_STACK(stack)\
    stack.pFirstStatement = NULL;\
    stack.pLastStatement = NULL;
```

2.4.2.2. INIT_CHAIN

```
#define INIT_CHAIN(Chain)\
    Chain.bIsApplied = false;
```

2.4.3. FREE

```
#define FREE(pFirst, p)\
    while(pFirst){\
        p = pFirst->pNext;\
        free(pFirst);\
        pFirst = p;\
    }\
    p = pFirst;
```

2.4.3.1. FREE_STACK

```
#define FREE_STACK(stack)\
    FREE(stack.pFirstStatement, pStatement)
```

2.4.4. NEW

2.4.4.1. NEW_PAGE

```
#define NEW_PAGE(pPage, iChainBasePage)\
    pPage = new Page;\
    pPage->iPage = iChainBasePage;\
    pPage->pFirstCard = NULL;\
    pPage->pLastCard = NULL;\
    pPage->pRootTmpVar = NULL;\
    pPage->pRootSrcVar = NULL;\
    pPage->pPrev = NULL;\
    pPage->pNext = NULL;
```


2.4.4.2. NEW_CARD

```
#define NEW_CARD(pCard, iChainBaseCard)\
    pCard = new Card;\
    pCard->iCard = iChainBaseCard;\
    pCard->iEntry = -1;\
    pCard->pFirstChain = NULL;\
    pCard->pLastChain = NULL;\
    pCard->pFirstUnit = NULL;\
    pCard->pLastUnit = NULL;\
    pCard->pPrev = NULL;\
    pCard->pNext = NULL;
```

2.4.4.3. NEW_ELEMENT

```
#define NEW_ELEMENT(pElement, Ele)\
    pElement = new Element;\
    pElement->pPrev = NULL;\
    pElement->pNext = NULL;\
    pElement->pFirstAttr = NULL;\
    pElement->pLastAttr = NULL;\
    pElement->pChildChain = NULL;\
    pElement->bIsChainBase = false;\
    pElement->Ele = Ele;\
    pElement->Ele.bIsAbsPosOrg = Ele.cAbsPos;\
    pElement->Ele.bIsChanged = FALSE;
```

2.4.4.4. NEW_CHAIN

```
#define NEW_CHAIN(pChain, targetEle)\
    pChain = new Chain;\
    pChain->bIsApplied = false;\
    NEW_ELEMENT(pChain->pChainBase, targetEle);\
    pChain->pFirstElement = NULL;\
    pChain->pLastElement = NULL;\
    pChain->pPrev = NULL;\
    pChain->pNext = NULL;
```

Macro NEW_CHAIN is used in private method NewChain, NewChildChain, NewDivChildChain.

2.4.4.5. NEW_CHILD_CHAIN

```
#define NEW_CHILD_CHAIN(pChain)\
pChain = new Chain;\
pChain->bIsApplied = false;\
pChain->pChainBase = NULL;\
pChain->pFirstElement = NULL;\
pChain->pLastElement = NULL;\
pChain->pPrev = NULL;\
pChain->pNext = NULL;
```

2.4.4.6. NEW_UNIT

```
#define NEW_UNIT(pUnit, pElement)\
pUnit = new Unit;\
pUnit->pElement = pElement;\
pUnit->pPrev = NULL;\
pUnit->pNext = NULL;
```

2.4.4.7. NEW_FAMILY

```
#define NEW_FAMILY(pFamily, pUnit)\
pFamily = new Family;\
pFamily->iFamilyId = pUnit->pElement->Ele.iFamilyId;\
pFamily->pFirstUnit = pFamily->pLastUnit = pUnit;\
pFamily->pPrev = pFamily->pNext = NULL;
```

2.4.4.8. NEW_VAR

```
#define NEW_VAR(pVar, sCurrFrame)\
pVar = new Var;\
pVar->iMaxFrame = 0;\
pVar->iMaxIFrame = 0;\
pVar->sFrame = sCurrFrame;\
pVar->bToOutput = false;\
pVar->pPrev = NULL;\
pVar->pNext = NULL;\
pVar->pFirstFrame = NULL;\
pVar->pLastFrame = NULL;\
pVar->pFirstIFrame = NULL;\
pVar->pLastIFrame = NULL;
```

2.4.5. APPEND

```
#define APPEND(pList, member, pItem, ItemName)
if(pList->member##pLast##ItemName)
    pList->member##pLast##ItemName->pNext = pItem;
else
    pList-> member##pFirst##ItemName=pItem;
pItem->pPrev = pList-> member##pLast##ItemName;
pItem->pNext = NULL;
pList->member##pLast##ItemName = pItem;
```

```
#define APPEND_M(pList, member, pItem, ItemName)\
if(pList->member##pLast##ItemName)\
    pList->member##pLast##ItemName->pNext = pItem;\
else\
    pList-> member##pFirst##ItemName=pItem;\
pItem->pPrev = pList-> member##pLast##ItemName;\
pItem->pNext = NULL;\
pList->member##pLast##ItemName = pItem;
```

2.4.5.1. APPEND_PAGE

```
#define APPEND_PAGE(this, pPage)\
APPEND_M(this, m_, pPage, Page)
```

2.4.5.2. APPEND_CARD

```
#define APPEND_CARD(pPage, pCard)\
APPEND(pPage, pCard, Card)
```

2.4.5.3. APPEND_ELEMENT

```
#define APPEND_ELEMENT(pChain, pElement)\
APPEND(pChain, pElement, Element)
```

2.4.5.4. APPEND_CHAIN

```
#define APPEND_CHAIN(pChain)\
APPEND_M(this, m_, pChain, Chain)
```

2.4.5.5. APPEND_CHAIN_TO_CARD

```
#define APPEND_CHAIN_TO_CARD(pCard, pChain)\
APPEND(pCard, pChain, Chain)
```

2.4.5.6. APPEND_DEL_CHAIN

```
#define APPEND_DEL_CHAIN(pChain)\
APPEND_M(this, m_, pChain, DelChain)
```

2.4.5.7. APPEND_ATTR

```
#define APPEND_ATTR(pElement, pAttr)\
APPEND(pElement, pAttr, Attr)
```

2.4.5.8. APPEND_UNIT

```
#define APPEND_UNIT(pCard, pUnit)
APPEND(pCard, pUnit, Unit);
```

2.4.5.9. APPEND_FAMILY

```
#define APPEND_FAMILY(pCard, pFamily)\
APPEND(pCard, pFamily, Family)
```

2.4.5.10. APPEND_UNIT_TO_FAMILY

```
#define APPEND_UNIT_TO_FAMILY(pCard, pUnit)\
bFamilyIsFound = false;\
for(pFamily = pCard->pFirstFamily; pFamily; pFamily = pFamily->pNext){\

    if(pFamily->iFamilyId == pUnit->pElement->Ele.iFamilyId){\
        APPEND(pFamily, pUnit, Unit);\
        bFamilyIsFound = true;\
        break;\
    }\
}\
if(bFamilyIsFound == false){\
    NEW_FAMILY(pFamily, pUnit);\
    APPEND_FAMILY(pCard, pFamily);\
    APPEND(pFamily, pUnit, Unit);\
}
```

2.4.6. STATEMENT

2.4.6.1. PUSH_STATEMENT

```
#define PUSH_STATEMENT(Stack, pStatement)\
if(Stack.pLastStatement)\
    Stack.pLastStatement->pNext = pStatement;\
else\
    Stack.pFirstStatement=pStatement;\
pStatement->pPrev = Stack.pLastStatement;\
pStatement->pNext = NULL;\
while(pStatement->pNext != NULL)\
    pStatement=pStatement->pNext;\
Stack.pLastStatement = pStatement;
```

2.4.6.2. POP_STATEMENT

```
#define POP_STATEMENT(Stack, pStatement)\
pStatement = Stack.pLastStatement;\
while(pStatement->bNewAction == false)\
    pStatement = pStatement->pPrev;\
Stack.pLastStatement = Stack.pLastStatement->pPrev;\
if(Stack.pLastStatement)\
    Stack.pLastStatement->pNext = NULL;\
else\
    Stack.pFirstStatement = NULL;
```

2.4.7. IS

2.4.7.1. IS_DESCENDANT

Decide whether Element Ele_2 is the descendant of Element Ele_1.

```
#define IS_DESCENDANT(Ele_1, Ele_2)\
    (Ele_1.iFamilyId == Ele_2.iFamilyId &&\
     Ele_1.sPath.substring(Ele_2.sPath)
```

2.4.7.2. IS_EQUAL

Decide whether Element Ele_2 equals Element Ele_1.

```
#define IS_EQUAL(Ele_1, Ele_2)\
    (Ele_1.iFamilyId == Ele_2.iFamilyId &&\
     Ele_1.sPath.equals(Ele_2.sPath))
```

2.4.8. INSERT

p1 and p2 are two pointers.

2.4.8.1. INSERT_BEFORE

Insert the structure instance pointed by p1 before the one pointed by p2.

```
#define INSERT_BEFORE(pFirst, p1, p2)
```

```
if(p2 == pFirst)\
```

```
    pFirst = p1;\
```

```
else\
```

```
    p2->pPrev->pNext= p1;\
```

```
\
```

```
p1->pPrev = p2->pPrev;\
```

```
p1->pNext=p2;\
```

```
p2->pPrev=p1;
```

2.4.8.2. INSERT_AFTER

Insert structure instance pointed by p2 after the one pointed by p1

```
#define INSERT_AFTER(pLast, p1, p2)
```

```
if(p1 == pLast)\
```

```
    pLast = p2;\
```

```
else\
```

```
    p1->pNext->pPrev= p2;\
```

```
p2->pNext = p1->pNext;\
```

```
p2->pPrev=p1;\
```

```
p1->pNext=p2;\
```

2.4.9. CUT

```
#define CUT(pFirst, pLast, pCurrent)\
if(pCurrent == pFirst){\
    pFirst = pCurrent->pNext;\
}\
else{\
    if(pCurrent == pLast){\
        pLast = pCurrent->pPrev;\
        pLast->pNext = NULL;\
    }\
    else{\
        pCurrent->pPrev->pNext = pCurrent->pNext;\
        pCurrent->pNext->pPrev = pCurrent->pPrev;\
    }\
}\
}
```

pCurrent->pPrev = pCurrent->pNext = NULL;

2.4.9.1. CUT_ELEMENT

```
#define CUT_ELEMENT(pFirst, pLast, pCurrent)\
    CUT(pFirst, pLast, pCurrent)\
    pCurrent->bIsChainBase = false;
```

2.4.10. REPLACE

```
#define REPLACE(pFirst, pLast, pOld, pNew)\
if(pOld == pLast){\
    pLast = pNew;\
}\
if(pOld == pFirst){\
    pFirst = pNew;\
}\
if(pOld->pPrev)\
    pOld->pPrev->pNext = pNew;\
if(pOld->pNext)\
    pOld->pNext->pPrev = pNew;\
pNew->pNext = pOld->pNext;\
pNew->pPrev = pOld->pPrev;\
pOld->pPrev = pOld->pNext = NULL;
```


2.5. Class Definition

Appendix B

```

class RuleGenerator{
private:
    Stack    m_redoStack;
    Stack    m_undoStack;
    Page     *m_pFirstPage;
    Page     *m_pLastPage;
    Chain    *m_pDelChain;
    Chain    *m_pFirstChain;
    Chain    *m_pLastChain;
    Chain    *m_pFirstDelChain;
    Chain    *m_pLastDelChain;
    bool m_bCanUndo, m_bCanRedo;

public:
    PageRule m_FirstRule;

private:
    void DestroyElement(Element *pElement);
    Element *DeleteElement(Chain *pChain, Element *pElement);
    void LocateElement(Chain *pChain, ElementInfo *pElementInfo);
    void NewChain(ElementInfo *pElementInfo, Statement *pStatement);
    Chain *DeleteChain(Chain *pChain, Card *pCard=NULL, int
mode=FROM_NOWHERE);
    void NewChildChain(Element *pParentElement, char cAction, Element
*pSourceElement);
    void UpdateChain(ElementInfo *pElementInfo, Statement *pStatement);
    void UpdateDelChain(TagId Ele, char cAction);
    void FilterDelChain();
    void AssemblyChain();
    void ParseFrame(Chain *pChain, Page *pPage);
    void OutputVar(ostream& xsltFile, Var *pVar, DOMString sFrom);
    void OutputChain(ostream& xsltFile, Card *pCard, Chain *pChain);
    Element *GetUnit(Card *pCard);

public:
    Statement();           //initialize the redo & undo stack(alloc memory for them)
    ~Statement();          //free the space

public:
    void PushStatement(char cAction, TagId sourceEle, TagId targetEle, bool bNewAction);
    void PushStatement(char cAction, TagId sourceEle, bool bNewAction);
    void PushStatement(TagId sourceEle, DOMString psNewAttrName[], DOMString
psNewAttrValue[], int iNumOfAttr, bool bNewAction);
    void PushStatement(TagId sourceEle, DOMString sNewText, bool bNewAction)
    void UndoStatement();
    void RedoStatement();
    bool CanRedo();
    bool CanUndo();
    void GenerateXSLT();
    char *SaveTempXSLT(int iPage, int iFrame);
    int ReloadXSLT(char *pszXsltFile, int iPage, int iFrame);
};

```